UNIVERSIDAD CARLOS III DE MADRID

Departamento de Ingeniería de Sistemas y Automática

ADAPTACIÓN DE ENTORNOS Y ELEMENTOS ROBÓTICOS CREADOS PARA LA PLATAFORMA DE SIMULACIÓN GAZEBO A OPENRAVE

Autora: Isabel García Molina

Tutora: Concepción Alicia Monje Micharet

Directora: Tamara Ramos Cambero

RESUMEN

El proyecto consiste en la adaptación de entornos de simulación de robots creados para la plataforma de simulación robótica Gazebo con el objetivo de trabajar con estos mismos entornos en simulaciones con OpenRAVE. A lo largo del documento se analiza y compara cómo se especifica un entorno de simulación en ambos programas y se detalla el proceso seguido para la conversión, incluyendo las herramientas y manipulaciones de fichero necesarias. Los entornos que se han migrado proceden de la Competición Robótica de DARPA. Como resultado de la adaptación, se obtienen entornos y elementos robóticos funcionales en OpenRAVE, con apariencia y propiedades similares a los originales en Gazebo. El proceso de adaptación descrito es generalizable a otros entornos, por lo que este documento puede resultar de utilidad como referencia para futuras conversiones.

ABSTRACT

This project aims to adapt certain simulation environments designed for the robotic simulation platform Gazebo in order to allow them to be used in OpenRAVE simulations. This document analyses and compares how a simulation environment is created for each program. Building on this knowledge, the conversion process is explained, including the required tools and file manipulations. The migrated environments are part of the DARPA Robotics Challenge. The resulting environments and robotics elements for OpenRAVE are fully functional, with features and appearance similar to the originals in Gazebo. The conversion process can be generalized to other environments, which renders this document useful for reference in future migrations.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE FRAGMENTOS DE CÓDIGO

ÍNDICE DE TABLAS

INTRODUCCIÓN

MOTIVACIÓN DEL PROYECTO Y OBJETIVOS

El proyecto surge por la necesidad de trabajar en la plataforma de simulación OpenRAVE con entornos creados para Gazebo.

En el departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III se quieren desarrollar algoritmos de locomoción y estabilidad para robots. Para poder estudiar el comportamiento del robot y mejorar su desempeño en estas habilidades es necesario simular el robot en un entorno que proporcione las pruebas adecuadas. Los entornos desarrollados para la Competición Robótica de DARPA (DRC) cumplen esta finalidad, ya que están orientados a evaluar, entre otras habilidades, la movilidad de los robots.

Dichos entornos están creados para su uso en Gazebo, mientras que el software de simulación empleado en la Universidad es OpenRAVE. El objetivo es, por tanto, adaptar los entornos de las pruebas de DRC al formato de OpenRAVE, para poder realizar las simulaciones con el robot en OpenRAVE en dichos entornos.

Derivado de este objetivo inicial concreto, se establece una finalidad más amplia: generalizar la conversión de entornos de Gazebo a OpenRAVE para distintos escenarios con elementos y características variados, que puede tener interés en trabajos futuros. Para ello se va a realizar un análisis en detalle del proceso de creación de entornos tanto en Gazebo como en OpenRAVE, poniendo especial atención a la relación entre ambos.

ORGANIZACIÓN DE CAPÍTULOS

En el Capítulo 1 se hace una introducción a la Competición Robótica de DARPA. Se ven someramente las pruebas que componen la competición y más en detalle los entornos que tienen interés para el proyecto. Se habla del simulador DRCSim que se emplea como plataforma de simulación en la competición y del robot Atlas desarrollado para participar en la misma.

El Capítulo 2 es un resumen de las características generales de los programas de simulación con los que se trabaja en el proyecto: Gazebo y OpenRAVE.

A lo largo del Capítulo 3 se analiza cómo crear entornos en Gazebo y en OpenRAVE. En la realización de este análisis se busca la correspondencia entre ambas plataformas en cada paso, ya que el objetivo final es hacer la conversión de una a otra. Este capítulo proporciona el marco teórico que ha servido de base para la realización de este proyecto.

El Capítulo 4 describe el proceso que se ha seguido en la práctica para adaptar los entornos de las pruebas de DRC, originalmente en Gazebo, a OpenRAVE. Se incluye toda la información relativa a las herramientas que se han empleado en la conversión.

En el Capítulo 5 se exponen los resultados del proceso de adaptación. Se presentan imágenes de los entornos tal y como están definidos inicialmente en Gazebo y de los obtenidos tras la conversión en OpenRAVE.

El Capítulo 6 recoge las conclusiones del proyecto, aportaciones y posibles trabajos futuros.

ESTADO DEL ARTE

La Robótica ha experimentado un gran desarrollo desde sus inicios con autómatas simples hasta los actuales robots capaces de desarrollar tareas complejas. Con la evolución de los robots aumentan las posibilidades pero también la complejidad de los mismos. Esto redunda en que resulta cada vez más difícil evaluar la actuación de los robots en situaciones reales, para probar su funcionamiento y continuar con el desarrollo del robot.

Para ayudar en el desarrollo del robot sin necesidad de recrear en la realidad escenarios complejos se recurre a plataformas de simulación de robots. Dichas plataformas permiten simular virtualmente tanto el robot como todo lo que lo rodea, simplificando la evaluación necesaria para el desarrollo del mismo. La utilización de simuladores proporciona además otras grandes ventajas frente a probar el robot físico en un entorno real: se reducen costes y tiempo, aumenta la flexibilidad y posibilidades de las pruebas que evalúan el robot y se minimizan los riesgos para robot físico.

Aunque en este proyecto se va a trabajar únicamente con los simuladores Gazebo y OpenRAVE, es importante tener presente que existen otras muchas herramientas disponibles para simulación de robots, que ofrecen distintas posibilidades. Algunas de las más relevantes (por su grado de difusión en la comunidad robótica) son [1]:

- ARGoS. Es una plataforma de simulación de robots de código abierto. Se orienta a la simulación de enjambres de robots. Permite utilizar diferentes motores físicos simultáneamente [2].
- V-Rep. Software para simulación robótica que se basa en una arquitectura de control distribuida que permite controlar cada elemento de manera individual, lo que lo hace muy versátil. Se emplea para desarrollar algoritmos de manera rápida [3].
- Webots. El simulador Webots se destina principalmente a enseñanza e investigación. Se trata de un software comercial propiedad de la empresa Cyberbotics Ltd [4].
- Robotran. Es un software simbólico que permite el modelado y análisis de Sistemas Multicuerpo (del inglés, MultiBody Systems) [5].

Aunque las ventajas de emplear un simulador son abrumadoras, la utilización de estas herramientas presenta también algunas limitaciones. En ocasiones puede tener interés trabajar con un robot en distintas plataformas, ya sea porque éstas proporcionan diferentes posibilidades de las que se quiere hacer uso, porque permiten evaluar distintos aspectos en la simulación, porque se quiere reducir la posibilidad de error del robot antes de pasar al modelo físico, o porque se quiere emplear un mismo robot o escenario en distintos proyectos en los que se utilizan diferentes plataformas. Sin embargo, debido a las diferencias en el funcionamiento de los distintos programas, un robot o un entorno que han sido creados para su simulación en una plataforma determinada no pueden utilizarse directamente en otra plataforma, es necesario realizar previamente una adaptación de los mismos. Este proyecto se centra en dicho proceso de integración entre las plataformas. Concretamente, en la migración de entornos de Gazebo a OpenRAVE.

Ya existen herramientas, de las que se habla en el Capítulo 3, que realizan la conversión automática de robots descritos en el formato de Gazebo al formato empleado en OpenRAVE, y viceversa. Sin embargo, aún no se ha creado ninguna aplicación que permita la adaptación de entornos y otros elementos robóticos de un programa a otro. Este documento pretende analizar las características y proponer un proceso de adaptación que permita utilizar los mismos entornos en ambos programas.

1 DARPA ROBOTICS CHALLENGE

DRC (por sus siglas en inglés, DARPA Robotics Challenge) es una competición organizada por DARPA [1] que pone a prueba la capacidad de los robots para afrontar desastres y prestar asistencia en condiciones de peligro. Los robots de los equipos participantes han de ser capaces de superar con relativo éxito una serie de tareas previamente definidas por DARPA orientadas a evaluar la actuación de los robots en diferentes circunstancias de emergencia [6].

1.1 OBJETIVOS

La competición surge como un plan estratégico de la agencia DARPA, con el objetivo de mejorar la tecnología para gestión de situaciones catastróficas que existe actualmente [7]. La competición se utiliza como herramienta para impulsar el desarrollo de robots humanoides que sean capaces de enfrentarse con éxito a desastres de diversa naturaleza, tanto de tipo natural como de origen humano [6].

1.2 PRUEBAS

Esta competición consta de una serie de pruebas o tareas. Dichas pruebas están escogidas por DARPA por su relevancia a la hora de evaluar la respuesta de los robots ante desastres y en condiciones desfavorables, y los entornos simulados recrean situaciones catastróficas reales [7], [8]. Cada prueba mide una serie de habilidades, de entre: movilidad, destreza manual, manipulación, fuerza, percepción y capacidad de decisión [9].

En la Figura 1.1 se puede ver un esquema de todas las pruebas preparadas para los entrenamientos de DRC (DRC Trials) que tuvieron lugar en diciembre de 2013. Las pruebas y habilidades evaluadas en cada una de ellas son las siguientes [9]:

1. Conducir y abandonar un vehículo. Supone un reto de destreza y fuerza.
2. Atravesar a pie un terreno rugoso y con desniveles. Requiere estabilidad y capacidad de identificar las rutas más seguras.
3. Limpiar el rellano de escombros. Precisa una amplia movilidad, equilibrio y fuerza.
4. Abrir una serie de puertas. Evalúa percepción y destreza.
5. Subir una escalera. Requiere equilibrio y fuerza para evitar una caída.
6. Derribar una pared. Mide la fuerza, destreza y percepción del robot en el uso de herramientas.
7. Utilizar una manguera de incendios. Percepción, destreza y fuerza.
8. Identificar y cerrar válvulas con fugas. Precisa de habilidades de percepción para identificar las válvulas que es necesario cerrar y destreza para manipularlas.

[1] DARPA (Defense Advanced Research Projects Agency) es una división del Departamento de Defensa de Estados Unidos. Se encarga de la investigación y desarrollo de nuevas tecnologías orientadas a la resolución de problemas reales [74].

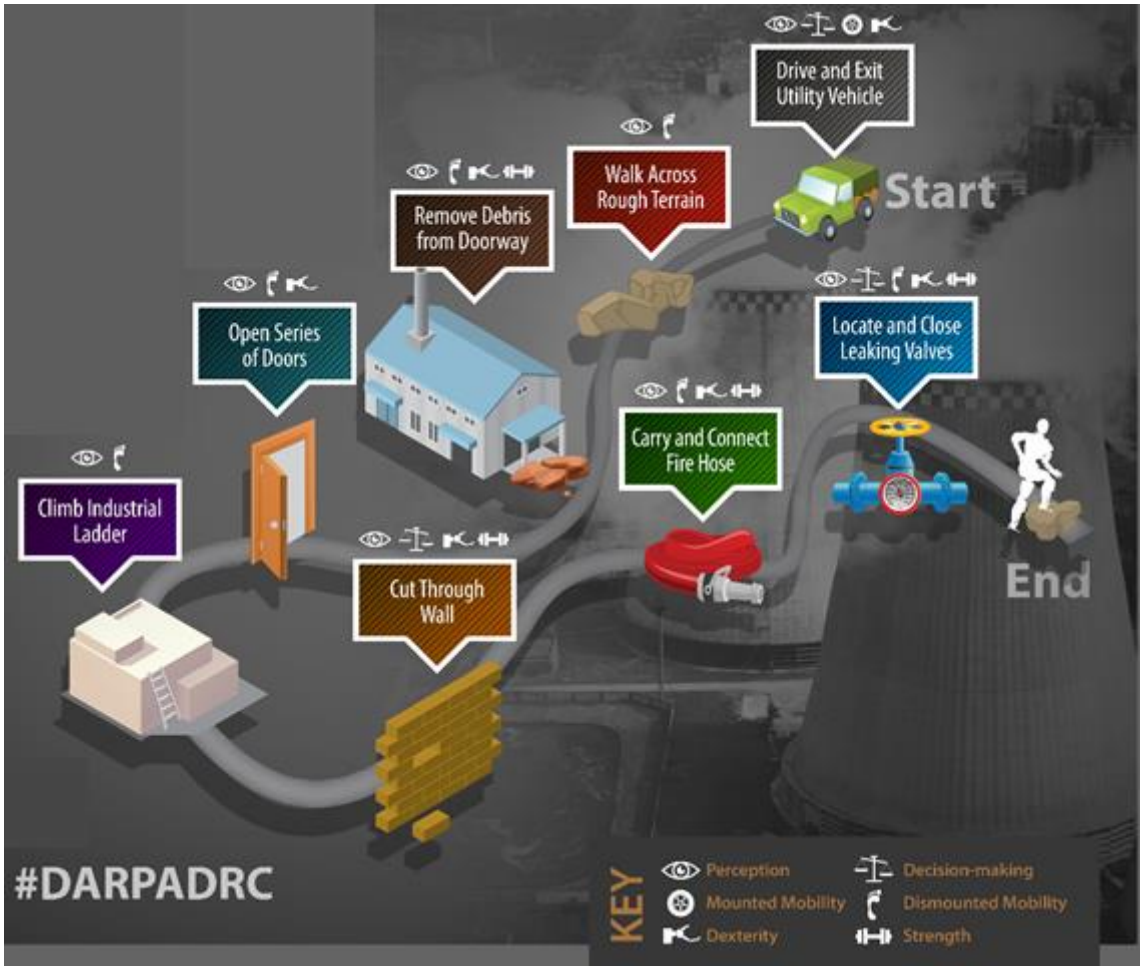

Figura 1.1. Pruebas DRC

1.3 ENTORNOS A MODELAR EN OPENRAVE

Este Proyecto está basado en las pruebas propuestas para los entrenamientos de la convocatoria del año 2013 descritas anteriormente.

Entre las pruebas planteadas, nos interesan aquellas que se centran en las habilidades de locomoción y estabilidad del robot. Según lo que se acaba de ver, éstas son: la tarea 2, que consiste en recorrer un terreno desigual, y la tarea 5, en que el robot ha de subir una escalera. Adicionalmente, nos interesa el escenario propuesto en la tarea 1, que está diseñado como un recorrido de obstáculos para un vehículo. En este caso se prescindirá del uso del vehículo que se incluye en la tarea, de esta manera, se podrá evaluar la capacidad para el cálculo de trayectorias del robot.

A continuación, se muestran algunas imágenes de los entornos de las tareas que se van a modelar en OpenRAVE, según aparecen el documento de descripción inicial de las pruebas de los entrenamientos de DRC [10]:

- Prueba 1. Conducción de un vehículo.

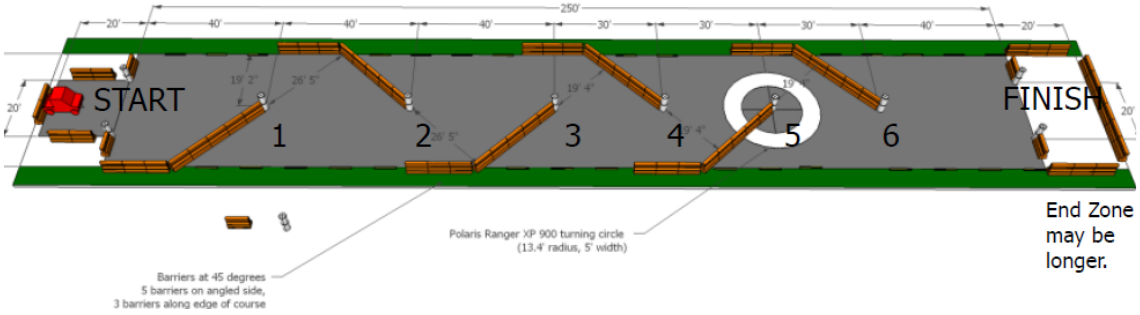

Figura 1.2. Prueba 1

- Prueba 2. Caminar en un terreno con desniveles.

Figura 1.3. Prueba 2

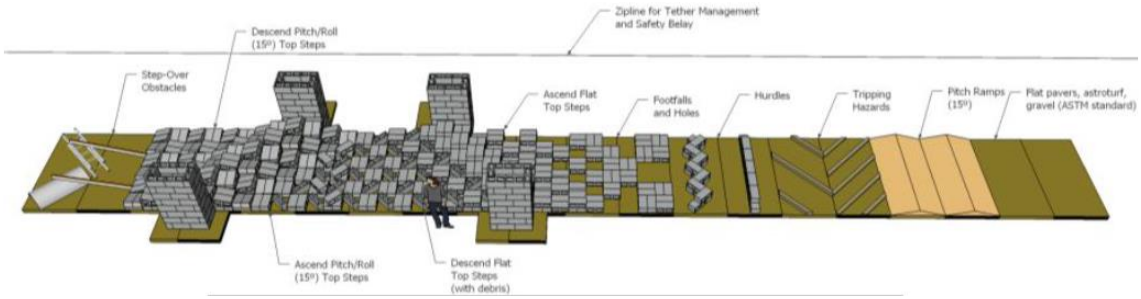

Figura 1.4. Prueba 2

- Prueba 5. Subir una escalera

Figura 1.5. Prueba 5

1.4 DRCSIM Y ATLAS

Para llevar a cabo esta competición, DARPA financia la creación de un simulador[2] (DRCSim) a través de la fundación OSRF[3]. Esta plataforma se pone a disposición de todos los participantes y permite simular los robots en los entornos específicos que se emplean en la competición. DRCSim modela con precisión la física y el comportamiento de los robots en el mundo real y permite cálculos en tiempo real [11]. Está basado en Gazebo[4], se construye a partir de éste incluyendo los entornos y modelos necesarios para simular todas las pruebas de DRC [12].

Además del simulador DRCSim, DARPA facilita a algunos de los equipos un robot para participar en la competición: Atlas, un robot humanoide creación de Boston Dynamics [13]. Atlas puede servir a los equipos como punto de partida para desarrollar su robot. Está orientado a la resolución de desastres, en consonancia con los objetivos de DRC. En la Figura 1.6 se resumen sus características más importantes [14].

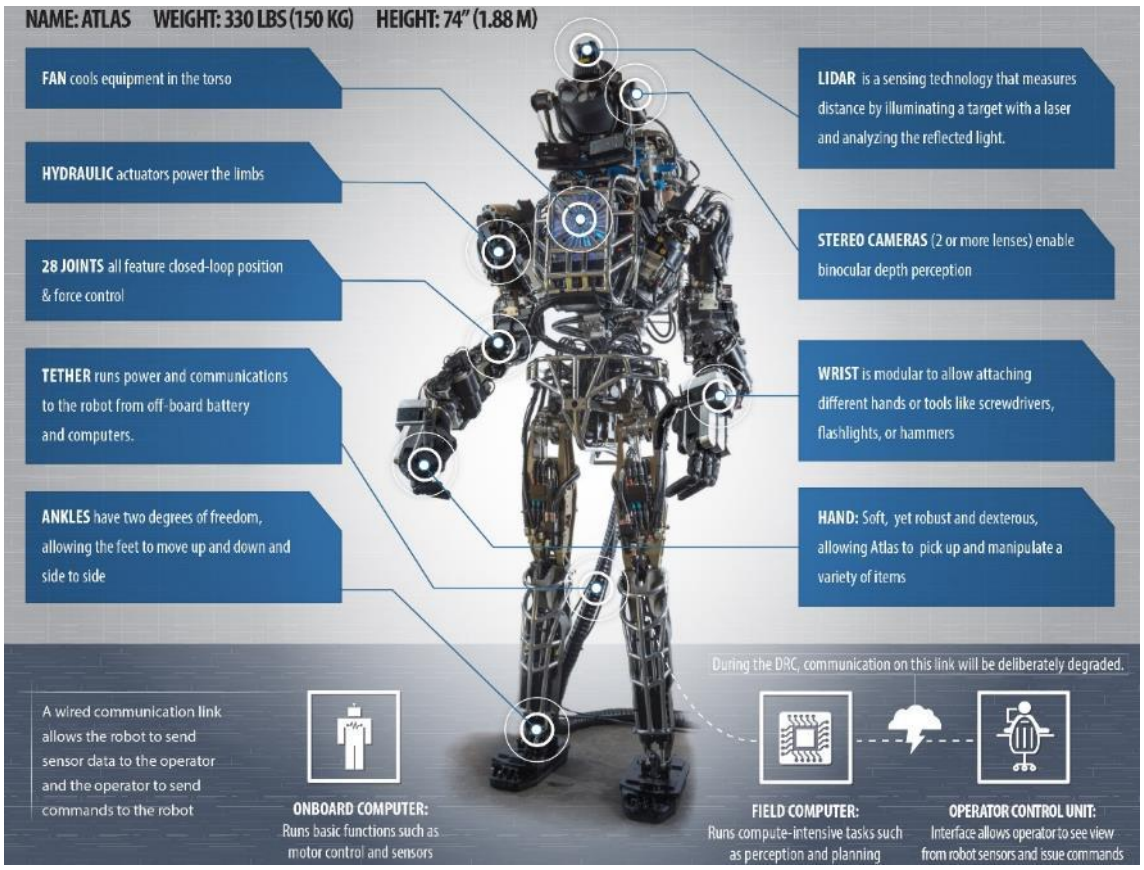

Figura 1.6. Atlas, un ejemplo de robot diseñado para la resolución de desastres

En la realización de este proyecto se parte del simulador DRCSim y de los modelos y entornos que se proporcionan con él, incluyendo a Atlas, y se adaptan para su uso en OpenRAVE.

---

[2] En el próximo capítulo se ve con más detalle lo que se entiende por simulador en Robótica.

[3] OSRF (Open Source Robotics Foundation) es una organización independiente y sin ánimo de lucro fundada por miembros de la comunidad robótica. El objetivo de la compañía es promover el desarrollo, distribución e integración de software de robótica de código abierto para su uso en investigación, enseñanza y desarrollo de productos [76].

[4] Gazebo es un programa de simulación de robots que se va a estudiar en los siguientes capítulos.

2 GAZEBO Y OPENRAVE: SIMULADORES DE ROBOTS

2.1 SIMULACIÓN EN ROBÓTICA

La simulación es el proceso de desarrollar un modelo virtual imitando la realidad. Esta definición se aplica tanto a Electrónica como a cualquier otro ámbito. Sin perder de vista la amplitud del concepto de simulación, cuando se hable de simulación en este proyecto, va a ser siempre referida al campo de la Robótica.

En concreto, un software de simulación de robótica o simulador sirve para diseñar y probar robots virtualmente, generalmente como paso previo a su construcción física. La mayoría de programas de simulación y, concretamente, los que se van a estudiar a continuación, simulan el robot dentro de un entorno, es decir, se incluye todo lo que rodea al robot, la escena donde va a actuar y desenvolverse, no sólo los objetos, sino también las propiedades físicas que rigen el entorno y afectan al robot.

Los simuladores permiten ahorrar costes y tiempo, a la vez que facilitan la tarea de realizar pruebas con el robot en diversos escenarios, más difíciles de recrear en la realidad que virtualmente mediante un programa. Además, evitan poner en riesgo el robot construido físicamente durante las pruebas. La versatilidad y la flexibilidad que brinda un simulador a la hora de hacer diferentes test con el robot es enorme, permite modificar de forma sencilla y sin coste el entorno, el robot, incluso el tiempo de la simulación, para adecuarlo a nuestras necesidades.

Existen numerosas alternativas a la hora de escoger un programa de simulación. Concretamente, en la elaboración de este proyecto se va a trabajar con:

- Gazebo, que es la plataforma en la que se apoya DARPA para definir las pruebas de la competición DRC; y
- OpenRAVE, el software utilizado en el departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III, para el que se quieren modelar los escenarios definidos en DRCSim en Gazebo.

A continuación se van a destacar algunos de los aspectos más relevantes de ambos.

2.2 GAZEBO. PLATAFORMA PARA DRCSIM

- Breve descripción

Gazebo es un programa de código abierto para simulación de robots. Permite simular un conjunto de robots, sensores y objetos en entornos tridimensionales, tanto interiores como exteriores, de manera precisa y eficiente. Integra un simulador de física del sólido rígido que permite generar relaciones físicas viables entre objetos [15], [16].

- Historia

Fue creado en el año 2002 en la University of Southern California por el Dr. Andrew Howard y su estudiante Nate Koenig. Este último ha continuado con el desarrollo del software en años

posteriores. A lo largo de los años, Gazebo se ha integrado con ROS[5], proporcionando una plataforma para simulación de robots muy potente y completa.

Como ya se ha visto, Gazebo es el programa base para el simulador DRCSim. La organización OSRF es actualmente la administradora del proyecto Gazebo, y se ha encargado de desarrollar Gazebo y adecuarlo para su uso en las competiciones de DARPA [16].

- Características

Algunas de las características más destacadas de Gazebo son: permite la simulación de propiedades dinámicas, integrando motores físicos de alto rendimiento: ODE, Bullet, Simbody y DART; proporciona gráficos en 3D de alta calidad, mediante la utilización de OGRE; es capaz de simular y captar información mediante sensores, cámaras, contacto u otros; funciona mediante la utilización de plugins para controlar robots, sensores y otros elementos del entorno; utiliza Python como lenguaje de programación para scripting; y, además, incluye modelos de robots y permite la creación de robots personalizados a través de ficheros SDF [16]. Algunas de estas características se analizarán con mayor detalle cuando se realice la comparación con OpenRAVE.

2.3 OPENRAVE

- Breve descripción

OpenRAVE (Open Robotics Automation Virtual Environment) es un software multiplataforma [17] de código abierto para simulación de robots. Su utilidad principal es la planificación de movimiento en aplicaciones con robots autónomos, orientadas al mundo real. Integra análisis y simulación tanto cinemática como geométrica [18].

- Historia

Es el resultado del trabajo de investigación de Rosen Diankov durante la realización de su tesis doctoral en el Instituto de Robótica de la Carnegie Mellon University de Pittsburgh, Pennsylvania. El desarrollo de este proyecto comienza en el año 2006 y continúa en la actualidad [19].

OpenRAVE surge por la necesidad de centralizar todos los procesos de alto nivel que deben realizarse para la simulación con robots. La planificación con robots en escenarios complejos es cada vez más crítica, y mediante este software se pretende ayudar a su simplificación. Para lograrlo, OpenRAVE se encarga de las tareas de detección de colisiones, cinemática de los elementos, dinámica del entorno, control de robots y scripting del entorno [20], de forma que el usuario sólo ha de preocuparse de la planificación y ejecución del código [21].

- Características

La característica y ventaja principal de OpenRAVE es que está basado en una arquitectura de plugins. Esta estructura permite conectar OpenRAVE fácilmente con otras plataformas y fomenta la cooperación de la comunidad robótica para compartir y comparar algoritmos [21]. De esta forma, OpenRAVE puede centrarse en los aspectos geométricos y dinámicos de la simulación y apoyarse en otros sistemas robóticos para el resto de tareas [19].

[5] ROS (Robot Operating System) proporciona un marco para el desarrollo de software de robótica. Consiste en un conjunto de librerías y herramientas para ayudar a desarrolladores de software en la creación de programas de robótica [73], [76].

Los motores físicos ODE y Bullet integrados en OpenRAVE proporcionan herramientas para el cálculo de colisiones, propiedades físicas y cinemáticas [20]. Para el renderizado en 3D, OpenRAVE utiliza la librería gráfica Coin3D [22].

En el siguiente apartado se estudian en mayor profundidad las características de Gazebo y OpenRAVE, para conocer las diferencias y similitudes que existen entre ambos.

3 ESTUDIO EN PARALELO DE GAZEBO Y OPENRAVE ORIENTADO A LA CREACIÓN DE ENTORNOS

A lo largo de este capítulo se van a estudiar las características de Gazebo y OpenRAVE, para analizar las diferencias y similitudes, y poder realizar la conversión entre ambos.

Se introduce el capítulo dando unas nociones sobre entornos y la creación de los mismos en los programas de simulación objeto de estudio. Partiendo de un conocimiento básico, en los siguientes apartados se estudian más en profundidad las diferencias entre los formatos utilizados en ambos programas. Al final del capítulo se hace una recopilación de los resultados obtenidos, resaltando las características que van a tener mayor relevancia para la conversión.

A la hora de comparar los programas, se estudian en primer lugar las características de Gazebo y a continuación se busca el paralelismo en OpenRAVE. Se escoge este orden teniendo en cuenta que el objetivo del proyecto es convertir un entorno definido para Gazebo en el equivalente legible por OpenRAVE, por lo que el primer paso es entender qué se está definiendo en Gazebo para después buscar la conversión adecuada a OpenRAVE.

3.1 INTRODUCCIÓN A LA CREACIÓN DE ENTORNOS. DEFINICIÓN DE ENTORNO

Como ya se ha comentado, se va a trabajar concretamente en la simulación de entornos y sus elementos. Por tanto, el primer paso es comprender qué se entiende por entorno en Robótica. En los siguientes apartados se va a estudiar cómo se define un entorno en un programa de simulación, cuál es su estructura básica, qué elementos lo componen y cuáles son sus características.

El **entorno** es el lugar donde se mueve el robot; está compuesto por todo lo que rodea e interacciona con el robot, incluyendo tanto objetos como características físicas que afectan al comportamiento del robot.

Dentro del entorno se pueden encontrar una serie de objetos, estáticos o dinámicos, que forman parte de la escena. Estos objetos se conocen como modelos. Un **modelo** es una entidad física dentro del entorno, en OpenRAVE se refiere también como **cuerpo cinemático**. Puede ser un robot, un sensor, una característica estática (como el suelo) o un objeto manipulable [23], [24].

Los **robots** son un tipo especial de modelo. La principal diferencia con otros modelos es que un robot posee mayores funcionalidades y es capaz de moverse en el entorno [25].

Un modelo, a su vez, está compuesto por uno o más **cuerpos** o **links**, cada uno de los cuales se define mediante una o varias **geometrías**. Para poder animar los modelos y mantener los links unidos entre sí se utilizan **articulaciones** o **joints** [26].

En la Figura 3.1 y Figura 3.2 se representa un entorno sencillo en OpenRAVE en el que se puede ver gráficamente cómo se distribuyen estos elementos en una simulación.

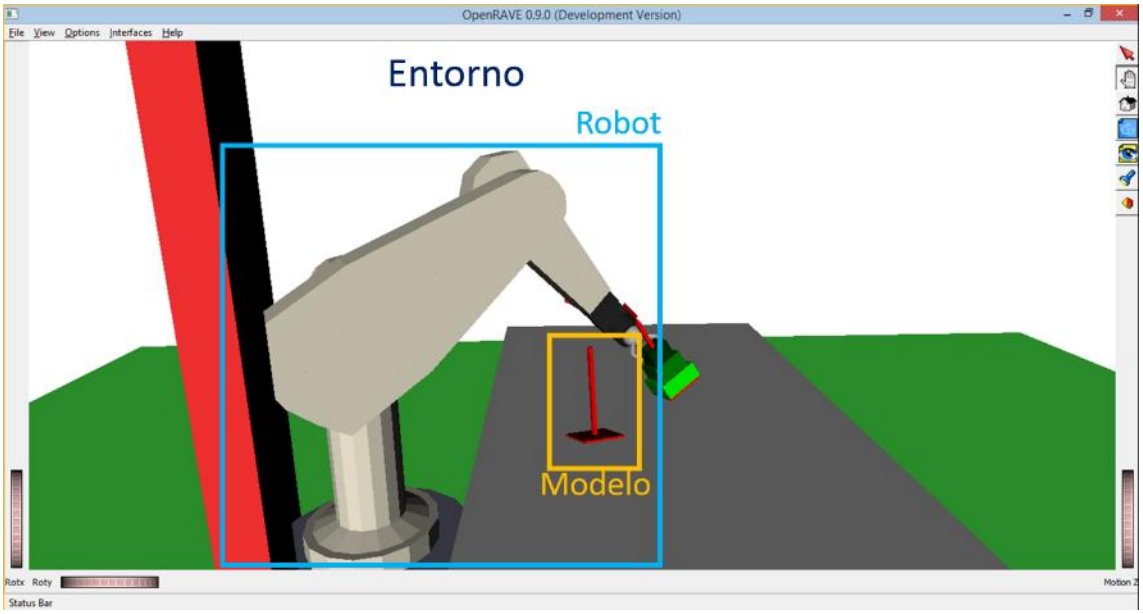

Figura 3.1. Modelos en un entorno en OpenRAVE

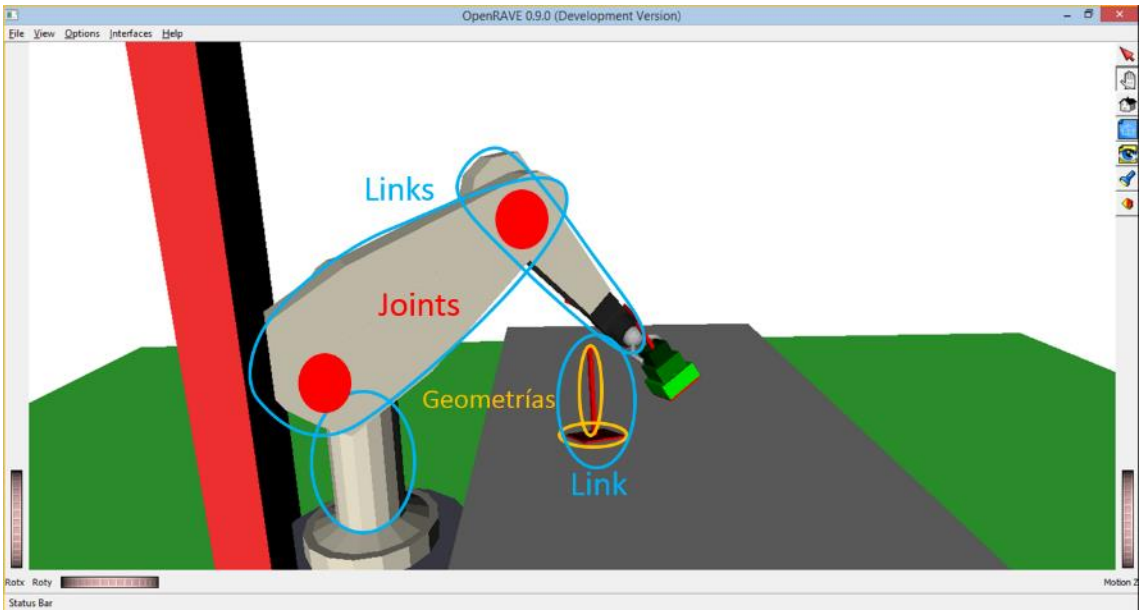

Figura 3.2. Elementos de un entorno en OpenRAVE

3.2 FORMATOS DE LENGUAJE

Para definir un entorno en un simulador se utilizan archivos que describen todos sus elementos y características. Para esta descripción ha de usarse un lenguaje comprensible para el programa. Los formatos de lenguaje empleados para definir el entorno están adaptados y son distintos para cada programa.

En ambos casos el formato de base para la definición de los escenarios y sus elementos es XML. A partir de éste se definen los formatos de lenguaje específicos para la funcionalidad buscada. En el caso de Gazebo, utiliza nuevos formatos basados en XML; OpenRAVE emplea un formato XML personalizado para adaptarlo a sus necesidades.

Se comienza dando una visión general de las características del lenguaje XML, común a ambos simuladores, y después se verán en detalle los formatos específicos utilizados en cada caso.

3.2.1 XML. Formato común

El formato XML (eXtensible Markup Language) es un formato flexible que permite el intercambio de enormes cantidades de datos a través de la Web y otros canales. Se trata de un lenguaje de marcas o etiquetado desarrollado por el World Wide Web Consortium[6] [27]. Su creación se inicia en 1996 y pretende ser una simplificación del lenguaje SGML[7], que facilite el intercambio de datos [28].

XML no es un lenguaje de programación, es un conjunto de reglas que permiten estructurar la información de forma ordenada e inequívoca. Se trata de un formato de texto con reglas de sintaxis muy rígidas. Utiliza *etiquetas* (indicándolas entre los signos "<", ">", por ejemplo, <*etiqueta*>) y estas etiquetas pueden tener asignados uno o varios *atributos* (expresados de la forma *atributo="valor"*). Las etiquetas y atributos sirven para estructurar y delimitar los datos y no tienen un significado definido previamente, sino que éste viene dado por el programa que interpreta el fichero XML [29].

Un formato basado en XML hereda la sintaxis de XML (uso de etiquetas y atributos) y la restringe en ciertos sentidos. Partiendo de las reglas de sintaxis de XML, define sus propias etiquetas y atributos, dando un significado concreto a la sintaxis y limitando el uso de etiquetas y atributos y los valores admitidos por éstos [29].

3.2.2 Gazebo

Gazebo utiliza en la definición de sus elementos formatos de lenguaje específicos derivados del formato universal XML.

- URDF (Universal Robot Description Format)

Es un formato de fichero XML utilizado en ROS para describir todos los elementos de un robot. Para el uso de un fichero URDF en Gazebo, se deben incluir etiquetas específicas para simulación [30], [31].

Los ficheros URDF son el formato estándar utilizado en ROS. Sin embargo, este formato carece de ciertas funcionalidades que han ido apareciendo al evolucionar el mundo de la robótica: mediante URDF sólo es posible especificar propiedades dinámicas y cinemáticas de un único robot aislado, no permite precisar la posición del robot, variables físicas como la fricción, ni definir elementos distintos de robots [30].

Para solucionar las carencias del formato URDF, se crea un nuevo formato: SDF [30].

- SDF (Simulation Description Format)

SDF es un formato basado en XML desarrollado para Gazebo. Pretende cubrir las deficiencias del formato URDF existente para creación de robots, permite una definición mucho más amplia del entorno, incluyendo todos los elementos y características que forman parte del mismo [30].

---

[6] W3C (World Wide Web Consortium) es la principal organización encargada de la creación de estándares para la web.

[7] SGML (Standard Generalized Markup Language) es un formato de lenguaje desarrollado en la década de 1980 y se utiliza principalmente en proyectos de documentación [29].

Un fichero SDF describe la física, la iluminación, el terreno, objetos estáticos o dinámicos, y robots articulados [32], [33].

Fue inicialmente creado para Gazebo, y a lo largo de los años se ha convertido en un formato estable y robusto que permite la definición de entornos y robots de una forma muy completa. Se trata de un formato expansible que permite añadir o modificar elementos de manera sencilla. Está basado en XML, lo que simplifica la actualización y migración a nuevas versiones, y es además autodescriptivo [33].

Todo el código de un archivo SDF está contenido dentro de una etiqueta **sdf**, que indica la versión utilizada:

```
<sdf version="1.5">
    <!-- código para creación del entorno -->
</sdf>
```

Fragmento de código 3.1. Cabecera de un archivo SDF

- Xacro

Xacro es un lenguaje XML basado en macros[8]. Es muy útil en la descripción de robots complejos u otros archivos XML extensos, ya que permite reducir la cantidad de código y describir los robots mediante ficheros XML más cortos y comprensibles. Además de proporcionar macros, el formato xacro permite la definición de constantes y de expresiones matemáticas. Todo ello ayuda enormemente en la simplificación y posterior comprensión del código [34], [35].

Más adelante se verán ejemplos del uso de xacros, al hablar de la adaptación del robot Atlas. En la definición de Atlas se utilizan ficheros con formato xacro, que ayudan en la simplificación de algunos elementos que repiten su estructura, como los dedos y las manos.

Un fichero *.xacro* puede convertirse a URDF mediante la utilización del programa xacro, que viene integrado con ROS, a través de la instrucción [35].

```
rosrun xacro xacro.py model.xacro > model.urdf
```

Esta herramienta resulta de gran utilidad para realizar la exportación automática del robot Atlas a OpenRAVE.

3.2.3 OpenRAVE

En OpenRAVE, el formato utilizado para la definición de todos sus robots y entornos es XML [36]. Adicionalmente, se pueden definir mediante archivos de COLLADA [37].

- XML

OpenRAVE almacena la información de entornos y robots en ficheros XML. Define sus propias etiquetas y jerarquías que personalizan el lenguaje XML y permiten describir el entorno y sus elementos y características de forma suficientemente detallada para la simulación [36].

Las etiquetas empleadas dentro del código del archivo XML dependen del elemento que se defina en el archivo: entorno, modelo o robot. Los distintos archivos y el uso de las etiquetas correspondientes se ven en el próximo apartado.

---

[8] Una macro es una plantilla para la ejecución de una secuencia de instrucciones de programación. Simplifica instrucciones complejas y permite automatizar tareas repetitivas y reutilizar código.

- COLLADA[9]

Para la definición de robots y entornos OpenRAVE admite además el uso de ficheros en formato COLLADA (de extensión *.dae*). Mediante COLLADA se pueden describir las propiedades geométricas, visuales y cinemáticas del robot y su entorno [37].

OpenRAVE va más allá de este formato, y define su propio formato derivado de COLLADA: el formato de extensión *.zae*. Dicho formato incluye una serie de extensiones específicas para robots: permite definir manipuladores, sensores, información sobre colisiones y parámetros específicos para planificación [38].

OpenRAVE permite la conversión fácilmente de un robot definido en formato XML a un formato *.zae*, para ello se utiliza la instrucción:

```
openrave -save myrobot.zae myrobot.xml
```

Además, OpenRAVE dispone de una herramienta en el paquete de ROS: `collada_urdf` (desarrollada por Rosen Diankov y Tim Field), que permite la conversión de archivos URDF al formato COLLADA y viceversa, de forma que se pueden cargar en OpenRAVE robots generados para otras plataformas o, al contrario, utilizar robots creados para OpenRAVE en formato COLLADA en otros programas [39]. Se verá esta funcionalidad más en detalle al estudiar la exportación del robot Atlas a OpenRAVE en el Capítulo 4.

3.3 Formatos de Archivos

En ambos programas es posible definir todos los elementos dentro del archivo de entorno, sin embargo, en ocasiones, puede tener interés definir en un fichero XML independiente robots y modelos de especial relevancia o que se repitan con frecuencia. El formato XML permite enlazar fácilmente ficheros XML entre sí, lo que facilita incluir en el entorno referencias a objetos y robots definidos en un archivo XML distinto.

Según el tipo de elemento que se quiera definir (entorno, modelo, robot) se utilizan diferentes archivos, con diferente nomenclatura y distintas características. En este apartado se exponen las características de estos archivos.

3.3.1 Archivos de entorno

Una simulación tiene siempre asociado un archivo de entorno. En el fichero de entorno se define la disposición de todos los elementos utilizados en la simulación: robots, sensores, objetos estáticos o dinámicos, características físicas. Además, puede servir para controlar parámetros de la simulación, como el paso de tiempo de simulación.

- Gazebo

En Gazebo estos ficheros se crean siguiendo el formato SDF y se nombran mediante la extensión ***.world*** [23]. Todo el contenido del entorno está dentro de la etiqueta **world**. En el Fragmento de código 3.2 se muestra la estructura básica de un fichero de entorno en Gazebo.

[9] El formato COLLADA se estudia en mayor profundidad en el apartado de Mallas Tridimensionales.

Nombre de archivo: drc_practice_task_1.world

```
<sdf version="1.4">
  <world name="trials_task_1">
    <!-- Descripción de elementos del entorno -->
  </world>
</sdf>
```

Fragmento de código 3.2. Archivo de entorno en Gazebo

- OpenRAVE

En OpenRAVE se utiliza el formato XML y la extensión ***.env.xml***. El contenido del entorno se define íntegramente dentro de la etiqueta **environment**, como se puede ver en el Fragmento de código 3.3.

Nombre de archivo: drc_practice_task_1.env.xml

```
<Environment>
  <!-- Descripción de elementos del entorno -->
</Environment>
```

Fragmento de código 3.3. Archivo de entorno en OpenRAVE

3.3.2 Modelos o cuerpos cinemáticos

Como ya se ha comentado, puede interesar definir modelos de cuerpos cinemáticos o robots de forma independiente al código principal del entorno, bien porque son modelos que se repiten a lo largo del código y se quiere evitar alargar el código innecesariamente; o bien porque se trata de un robot o un cuerpo complejo o de especial relevancia y tiene interés el tratarlo de forma separada. Para estos casos se crean archivos independientes que definen el modelo de manera aislada.

Los archivos que describen modelos son útiles para reutilizar modelos y simplificar los archivos de entorno. Una vez creado un archivo de este tipo, se pueden incluir en otros ficheros utilizando la sintaxis adecuada.

- Gazebo

Los archivos de modelos en Gazebo se definen con el mismo formato SDF que se emplea para los archivos de entorno.

Los modelos se estructuran en carpetas. Todos los ficheros necesarios para la definición del modelo se incluyen dentro de una carpeta con el nombre del modelo. En esta carpeta se encuentran los archivos: ***model.sdf***, que define en formato SDF la estructura del modelo; y *config.sdf*, que contiene metadatos sobre el modelo (nombre del modelo, autor, descripción del modelo). En los casos en que es necesario, se hallan también las carpetas *materials* y *meshes*, que contienen información relativa a las mallas 3D que forman parte del modelo [40]. Por ejemplo, la estructura que se sigue para definir un modelo llamado *wood_slats* es la siguiente:

- *wood_slats*: directorio para el modelo wood_slats.
  - *model.config*: metadatos sobre el modelo.
  - *model.sdf*: descripción del modelo en SDF.
  - *meshes*: directorio con los ficheros de mallas 3D asociados.
  - *materials*: directorio con texturas y plugins asociados a los modelos 3D.

Dentro del fichero *model.sdf*, en que se describe el modelo, se utiliza la etiqueta **model**. La estructura básica para el mismo modelo de ejemplo se presenta en el Fragmento de código 3.4.

```
<sdf version="1.5">
  <model name="wood_slats">
    <!-- Descripción del modelo -->
  </model>
</sdf>
```

Fragmento de código 3.4. Archivo de modelo en Gazebo

Para incluir un modelo definido de esta forma en el entorno, se referencia en el código mediante las etiquetas **include**, que indica que se va a añadir un fichero externo, y **uri**, que identifica la dirección del modelo. Siguiendo el ejemplo del modelo *wood_slats*, el código adecuado para incluirlo es el indicado en el Fragmento de código 3.5.

```
<include>
  <uri>model://wood_slats</uri>
</include>
```

Fragmento de código 3.5. Referencia a un modelo en Gazebo

- OpenRAVE

En OpenRAVE se utilizan archivos XML para describir los modelos o cuerpos cinemáticos. Estos ficheros tienen la extensión ***.kinbody.xml*** y se definen mediante la etiqueta **KinBody**, que contiene toda la descripción del modelo. Para el mismo modelo *wood_slats* que se ha tomado de ejemplo para Gazebo, la estructura del archivo sería:

Nombre de archivo: wood_slats.kinbody.xml

```
<KinBody>
  <!-- Descripción del modelo -->
</KinBody>
```

Fragmento de código 3.6. Archivo de modelo en OpenRAVE

Si los modelos tienen mallas 3D asociadas, los ficheros correspondientes se encuentran habitualmente en la carpeta *meshes*, la cual se halla en el mismo directorio que el fichero *.kinbody.xml*.

Los modelos en OpenRAVE se referencian en el entorno mediante el atributo **file**. Para incluir el modelo *wood_slats* en el entorno, se emplea el Fragmento de código 3.7.

```
<KinBody file="models/wood_slats.kinbody.xml">
    <!-- transformaciones aplicadas al modelo -->
</KinBody>
```

Fragmento de código 3.7. Referencia a un modelo en OpenRAVE

3.3.3 Robots

Un robot es un tipo especial de modelo. Como tal, la creación y posterior inclusión de archivos de robots en el entorno es muy similar a la descrita para modelos.

- Gazebo

La principal diferencia entre archivos de robots y modelos en Gazebo es que un robot se puede definir tanto en formato SDF como URDF. La estructura de carpetas para definir un robot sería la misma que para un modelo, pero el archivo que define la configuración del robot puede ser ***model.sdf*** o ***model.urdf***.

En el código de un archivo de robot, se utiliza la etiqueta **robot**, como en el Fragmento de código 3.8.

```
<robot name="drc_skeleton">
    <!-- descripción de todos los
    links y joints del robot -->
</robot>
```

Fragmento de código 3.8. Archivo de robot en Gazebo

Un robot se puede incluir en el entorno referenciándolo en el código del archivo de entorno (como hemos visto para los archivos de modelos) o como parte de las instrucciones de simulación.

- OpenRAVE

Un archivo de robot en OpenRAVE se nombra mediante la extensión ***.robot.xml*** y se define mediante la etiqueta **robot**, como se ve en el Fragmento de código 3.9.

```
<robot name="sandia_hands">
    <!-- descripción links y joints del robot -->
</robot>
```

Fragmento de código 3.9. Archivo de robot en OpenRAVE

También se pueden definir robots en formatos *.dae* y *.zae*, pero este proyecto se centra en la sintaxis para ficheros XML.

Para incluir un robot en el entorno se indica en el código de la misma forma que ya se ha visto para los archivos de modelos. También es posible incluirlo mediante scripting al lanzar la simulación (como se verá en el Capítulo 5).

3.3.4 Mallas Tridimensionales

Cuando se quiere representar un modelo utilizando geometrías complejas (distintas de cajas, cilindros y esferas) o incluir texturas, se define la geometría del modelo utilizando mallas tridimensionales[10]. Las mallas permiten añadir realismo a un modelo o robot. Se crean mediante programas de modelado en 3D y se almacenan en ficheros independientes, generalmente con formato de texto.

A continuación se va a hablar de los formatos de mallas 3D que admiten Gazebo y OpenRAVE y de cómo se realiza la importación en cada simulador.

- Gazebo

En el caso de Gazebo, los ficheros que definen las mallas deben estar en formato STL o COLLADA, siendo COLLADA el formato recomendado [26].

- Formato COLLADA

COLLADA (COLLAborative Design Activity) es un formato basado en XML de código libre creado para definir modelos 3D [41].

Se crea como respuesta a la necesidad de un formato estándar, que pueda utilizarse en las distintas plataformas que trabajan con objetos tridimensionales, para resolver problemas de compatibilidad entre formatos [42]. El consorcio Khronos Group se encarga actualmente del desarrollo y mantenimiento de COLLADA [41].

[10] Una malla tridimensional consiste en una superficie definida a partir de triángulos, mediante ternas de coordenadas correspondientes a los vértices y vectores normales que dan la orientación.

COLLADA permite definir geometrías de malla, transformaciones de los elementos, texturas, materiales, animación, cinemática e incluso propiedades físicas de las escenas representadas [41]. La extensión de un archivo en formato COLLADA es *.dae*.

- Formato STL

Se trata de un formato para representación de modelos 3D en que se prescinde de las texturas y otros atributos, sólo permite definir geometrías y colores sólidos. Muy útil para poder realizar cálculos de forma rápida cuando no se tiene tanto interés en la representación visual [43]. Los ficheros en formato STL tienen la extensión *.stl*.

- Asociar mallas 3D a un modelo

Dentro de la etiqueta **uri** se indica el directorio en que se encuentra la malla. Por ejemplo:

```
<uri>model://drc_practice_ladder/meshes/ladder.dae</uri>
```

Fragmento de código 3.10. Importación de malla 3D en Gazebo

- OpenRAVE

En OpenRAVE, los formatos admitidos para la definición de mallas 3D en un entorno son el formato VRML y Open Inventor [36]. COLLADA se puede emplear para definir robots completos, pero no en la descripción de geometrías de un modelo.

- Formato Open Inventor

Es una herramienta orientada a la creación de objetos tridimensionales. Está basado en una base de datos de entornos que simplifica enormemente la programación de gráficos en 3D, incluye objetos, texto, materiales, iluminación, visualizadores 3D, etc. Define un formato de fichero estándar para el intercambio de archivos 3D que ofrece una solución a los problemas de programación con gráficos interactivos. Se trata de un proyecto de software libre desarrollado por SGI[11] que se utiliza con OpenGL [44]. Los archivos Open Inventor tienen la extensión *.iv*.

- Formato VRML

VRML (Virtual Reality Modeling Language) es un formato de fichero diseñado para su uso en páginas web. Se trata del primer formato 3D basado en web, y soporta geometrías y animaciones en 3D. El consorcio Web3D Consortium se encarga actualmente del desarrollo de este formato, que ha evolucionado al formato X3D (Extensible 3D) [45]. La extensión asociada a los archivos en formato VRML es *.wrl*.

- Asociar mallas 3D a un modelo

Una malla se incluye mediante las etiquetas **render**, o **data**, o **collision**, según el papel que tengan en el modelo:

```
<render>meshes/ladder.iv</render>
```

Fragmento de código 3.11. Importación de malla 3D en OpenRAVE

3.4 Elementos del Entorno

En este apartado y los dos siguientes, se va a estudiar cómo se definen en el código del archivo de entorno los elementos y características necesarios, tanto en Gazebo como en OpenRAVE. La estructura que se va a seguir para realizar la comparación es siempre la misma.

[11] SGI (Silicon Graphics, Inc.) es una compañía que desarrolla herramientas informáticas de alto rendimiento [72].

En primer lugar, se recuerda brevemente qué papel tiene el elemento a analizar. Seguidamente, se explica cómo es la definición de este elemento o característica en cada formato, primero en Gazebo, luego en OpenRAVE. Por último, se comparan fragmentos de código equivalentes para cada programa y se resumen las diferencias y similitudes. Se concluye con indicaciones para realizar la adaptación de Gazebo a OpenRAVE.

3.4.1 Introducción. Jerarquía de Elementos en un Archivo de Entorno

A continuación se va a ver cómo se define cada uno de los elementos principales de un entorno, tanto en Gazebo como en OpenRAVE. En primer lugar se recuerda la relación jerárquica entre los elementos de un entorno.

El primer elemento del que van a derivar todos los demás es el **entorno**. Un entorno se compone habitualmente de una serie de declaraciones de **modelos**, que pueden ser objetos que forman parte del entorno o **robots**. Un modelo está formado por cuerpos sólidos o **links**, que pueden estar unidos mediante **articulaciones**. Dentro de cada cuerpo, se definen las características **geométricas** del mismo.

La estructura de los elementos al crear un archivo de entorno se refleja en la Figura 3.3.

```
Entorno

    Modelo1
        Link1
            Geometría1
            Geometría2
            ...
        Link2
        ...

    Modelo2
    ...

    Robot
        Link1
        ...
        Articulación1
        ...
```

Figura 3.3. Jerarquía de elementos en un entorno

En los siguientes apartados se estudia en profundidad cómo se define cada uno de estos elementos en el formato XML propio de cada simulador.

3.4.2 Entorno

El elemento principal en el que se incluyen todos los demás.

- Gazebo

Un archivo de entorno en Gazebo está indicado en el código mediante la etiqueta **world**, que contiene todo el código referido a los elementos y propiedades del entorno.

La etiqueta **world** admite el atributo **name**. Mediante este atributo se puede asignar al entorno creado el nombre que queramos.

- OpenRAVE

Un archivo de entorno en OpenRAVE se define mediante el uso de la etiqueta **Environment**, con todo el código que define el entorno y sus elementos dentro de esta etiqueta.

En OpenRAVE, en oposición a Gazebo, la etiqueta **Environment** que define un escenario no admite atributos, por tanto, en la adaptación a OpenRAVE se pierde el nombre que tienen los entornos en Gazebo.

- Comparación de códigos

Gazebo	OpenRAVE

`<world name="trials_task_1">`
`  <!-- Elementos entorno -->`
`</world>`
	`<Environment>`
	`  <!-- Elementos entorno -->`
	`</Environment>`

Tabla 3.1. Comparación de código de entorno

Conclusiones:

- La etiqueta **world** en Gazebo se sustituye por **Environment** en OpenRAVE.
- **world** dispone del atributo **name**, mientras que **Environment** no tiene atributos.

3.4.3 Modelos

Un modelo representa un objeto del entorno. Los modelos pueden estar incluidos en el código principal del entorno, o estar definidos en un fichero independiente. Se va a ver cómo se define un modelo y cómo se incluye en el entorno un modelo externo.

- Gazebo

El inicio de la definición del modelo se marca mediante la etiqueta **model**, y todo el código que se refiere al modelo ha de estar dentro de esta etiqueta.

En Gazebo, para hacer referencia a un fichero, se indica que se va a incluir información contenida en un fichero externo mediante la etiqueta **include**. Dentro de esta etiqueta, mediante la etiqueta **uri,** se indica el directorio en que se encuentra el modelo. En el caso de Gazebo, la dirección para un modelo corresponde a una carpeta, y dentro de ésta están los ficheros necesarios para cargar el modelo: *model.sdf*, *config.sdf*, como se ha visto en el apartado de Formatos de Archivos.

- OpenRAVE

En el código del archivo éste se define mediante la etiqueta **KinBody**, y toda la definición del modelo ha de estar dentro de esta etiqueta.

Para incluir un modelo definido en un archivo externo en OpenRAVE, se utiliza el atributo **file** de la etiqueta **KinBody**. El valor que se da a este atributo es la dirección del fichero a cargar, que será un archivo de tipo *.kinbody.xml*.

- Comparación de códigos

Gazebo	OpenRAVE

`<model name="ground_plane">`
`  <!-- Descripción modelo -->`
`</model>`
	`<KinBody name="ground_plane">`
	`  <!-- Descripción modelo -->`
	`</KinBody>`

Tabla 3.2. Comparación de código de modelo

Referencia a un modelo externo:

Gazebo	OpenRAVE

`<include>`
`  <uri>model://ground_plane</uri>`
`</include>`
	`<KinBody name="Floor"`
	`  file="models/ground_plane.kinbody.xml"/>`

Tabla 3.3. Comparación de código para referenciar un modelo

Conclusiones:

- La etiqueta **model** se sustituye por la etiqueta **KinBody**.
- Ambas etiquetas admiten el atributo **name**.
- La etiqueta **uri** en Gazebo apunta a la carpeta en que se encuentra el fichero *model.sdf*, mientras que el atributo **file** en OpenRAVE apunta directamente al fichero *.kinbody.xml*.

3.4.4 Robots

Un robot es un tipo especial de modelo o cuerpo cinemático. La definición de robots es similar a la de otro modelo del entorno, con algunas diferencias.

- Gazebo

Un robot se describe con la misma etiqueta **model** que se emplea en la definición de modelos. Los robots se caracterizan por estar formados mediante links unidos por articulaciones, para crear estos elementos se emplean las etiquetas **link** y **joint**, que se verán más adelante.

- OpenRAVE

La etiqueta **robot** engloba toda la descripción del robot en OpenRAVE. Dentro de esta etiqueta se define la etiqueta **KinBody**, que indica que se trata de un modelo o cuerpo cinemático del entorno. El robot está compuesto por una serie de links (**Body**) y articulaciones (**Joints**) que permiten el movimiento y mantienen los links unidos entre sí.

- Comparación de códigos

Gazebo	OpenRAVE

`<model name="robot1">`
`  <link>`
`  ...`
`  <joint>`
`  ...`
`</model>`
	`<Robot name="robot1">`
	`  <KinBody>`
	`  <Body>`
	`  ...`
	`  <Joint>`
	`  ...`
	`  </KinBody>`
	`</Robot>`

Tabla 3.4. Comparación de código de robot

Conclusiones:

- Un robot en Gazebo se declara mediante la etiqueta **model**. En OpenRAVE se utiliza la etiqueta **robot** específica y, dentro de la misma, la etiqueta **KinBody**.

3.4.5 Cuerpos o Links

Unidad básica de que se compone un modelo o cuerpo cinemático.

- Gazebo

Se define mediante la etiqueta **link**, que admite el atributo **name**. Dentro de esta etiqueta se describen las geometrías y transformaciones asociadas al cuerpo.

- OpenRAVE

Un cuerpo en OpenRAVE se define mediante la etiqueta **Body**, la cual admite los atributos **name** y **type**. El uso del atributo **type** se ve más adelante al estudiar las propiedades cinemáticas de los elementos.

- Comparación de códigos

Gazebo	OpenRAVE

`<link name="link">`
`  <!-- Geometría/Transformaciones -->`
`</link>`
	`<Body name="link" type="static">`
	`  <!-- Geometría/Transformaciones -->`
	`</Body>`

Tabla 3.5. Comparación de código de links

Conclusiones:

- La etiqueta **link** se sustituye por la etiqueta **Body**.
- En ambos simuladores se utiliza el atributo **name**, con el mismo significado.
- En OpenRAVE **Body** lleva asociado el atributo **type**, mientras que esta característica en Gazebo se indica en una etiqueta distinta.

3.4.6 Geometrías

Cada cuerpo se compone de una geometría o serie de geometrías, que pueden ser geometrías sencillas (como un cilindro o una esfera) o geometrías complejas definidas mediante una malla de triángulos.

En primer lugar, hay que entender un aspecto importante para clasificación de geometrías. Por un lado, la geometría afecta al aspecto visual del entorno; por otro lado, la geometría de los elementos determina también las colisiones. Teniendo en cuenta estas características, la geometría de un cuerpo se puede dividir en geometría visual y geometría de colisión.

Un cuerpo puede utilizar la misma representación para su geometría visual y de colisión, sin embargo, no siempre coinciden. Una buena práctica es utilizar geometrías simples para representar la colisión de los elementos, que pueden tener asociadas geometrías complejas como parte visual. Esto simplifica mucho los cálculos de colisiones, y mejora el rendimiento del simulador, sin detrimento de la estética del entorno.

En este apartado se habla de cómo definir geometrías según su parte visual y de colisión, más adelante se desarrollan las distintas formas geométricas y sus propiedades.

- Gazebo

Las geometrías en Gazebo se definen, según la clasificación que se acaba de ver, mediante dos etiquetas: **visual** y **collision** [26].

La etiqueta **collision** contiene la geometría que se emplea para el cálculo de colisiones. No se tiene en cuenta para la representación gráfica. Al contrario que para **collision**, las geometrías definidas dentro de la etiqueta **visual** sólo se emplean para la representación de los elementos en la interfaz gráfica. Para definir un elemento que se incluya en el cálculo de colisiones y tenga representación visual es necesario especificar su geometría dentro de ambas etiquetas.

Las características de la geometría se definen dentro de la etiqueta **geometry**, hija de la etiqueta **collision** o **visual** según corresponda.

El Fragmento de código 3.12 es un ejemplo del uso en Gazebo de estas etiquetas para definir las características geométricas de un cuerpo:

```
<model name="ramp_0">
  <pose>-2 0 0 0 0 0</pose>
  <static>true</static>
  <link name="link">
    <collision name="incline_collision">
      <geometry>
        <!-- Geometría de colisión -->
      </geometry>
    </collision>
    <visual name="incline_visual">
      <geometry>
        <!-- Geometría visual -->
      </geometry>
    </visual>
  </link>
</model>
```

Fragmento de código 3.12. Geometrías visual y de colisión en Gazebo

- OpenRAVE

En OpenRAVE, la geometría de un elemento se define dentro de la etiqueta **Geom**. Existen varias combinaciones posibles de geometrías visuales y de colisión, que se traducen a OpenRAVE de distintas formas. A continuación se exponen los casos más representativos y habituales.

- Caso 1. Misma geometría visual y de colisión

Mediante la etiqueta **Geom** no es necesario diferenciar parte visual y de colisión (aunque sí es posible). Una definición sencilla de una geometría de caja sería:

```
<Geom name="incline" type="box">
    <extents>.315552178 1.219198095 .05</extents>
</Geom>
```

Fragmento de código 3.13. Geometría simple en OpenRAVE

- Caso 2. Distinta geometría visual y de colisión

También es posible especificar las geometrías de forma independiente para la parte visual y para las colisiones, mediante el uso de etiquetas y atributos de **Geom**. Se hace de distintas formas según las razones por las que se quiera hacer esta separación.

- Caso 2.1. Geometría visual mediante malla tridimensional y colisión simple

Como ya se ha argumentado, es recomendable utilizar geometrías simples para el cálculo de colisiones, aun cuando se empleen geometrías complejas como representación visual. Por tanto, la primera razón para definir la geometría visual y de colisión de un cuerpo por separado es poder utilizar una definición distinta para cada caso: geometrías primitivas para calcular las colisiones, y mallas 3D para la representación gráfica. Esta distinción es posible en OpenRAVE mediante el uso de la etiqueta **render**. La etiqueta **render** se incluye dentro de **Geom** y sirve para referenciar el modelo 3D que se va a emplear como geometría visual del cuerpo.

Cuando se hace esta diferencia entre geometría visual y de colisión en OpenRAVE, se pueden dar dos casos: que la geometría visual corresponda con una única geometría primitiva de colisión (por ejemplo, una rueda puede simplificarse mediante un cilindro de las dimensiones adecuadas), o que el modelo visual esté representado por varias geometrías simples (una escalera correspondería con una serie de geometrías en forma de caja representando los escalones y cilindros para las barandillas). Se estudia cada caso por separado ya que el código a emplear es distinto.

- Caso 2.1.1. Una única geometría de colisión simple

Cuando una única geometría visual corresponde unívocamente con una geometría de colisión se incluye la definición de ambas partes dentro de la misma etiqueta **Geom**. Mediante la etiqueta **render** se indica que la parte visual está representada por el modelo 3D referenciado y que el resto del contenido de la etiqueta **Geom** que se refiere a la forma geométrica sólo se usa para cálculo de colisiones. Un ejemplo de cómo sería un cuerpo definido de esta forma:

```
<Body name="front_left_wheel" type="dynamic">
    <Geom type="cylinder">
        <!-- Geometría visual -->
        <render>meshes/polaris/Wheel_Front_Left.iv</render>
        <!-- Geometría de colisión -->
        <radius>0.3175</radius>
        <height>0.2794</height>
    </Geom>
</Body>
```

Fragmento de código 3.14. Geometría visual compleja y colisión simple en OpenRAVE

- Caso 2.1.2. Geometría de colisión formada por varias geometrías simples

En el segundo caso, hay varias geometrías de colisión que corresponden al mismo modelo visual. Es necesario utilizar etiquetas **Geom** independientes para cada geometría, ya que cada etiqueta **Geom** sólo admite un único elemento de colisión. Para indicar que se trata de geometrías de colisión, sin representación visual, se hace uso del atributo **render**, asignándole el valor **false** (este atributo admite los valores **true**, que es el valor por defecto, y **false**). La parte visual se define en un elemento **Geom** independiente, mediante la etiqueta **render**, igual que se hacía en el caso anterior. Nótese que el hecho de no especificar otra geometría de colisión en la etiqueta **Geom** no significa que no la haya. Si no se define explícitamente, OpenRAVE interpreta que la geometría de colisión es igual a la definida en la etiqueta **render**. Lo más aproximado que se puede hacer para eliminar la geometría de colisión en este caso es representarla mediante una geometría primitiva de tamaño singular (por ejemplo, una esfera de radio cero).

Se puede ver un ejemplo del uso del atributo **render** en la descripción de la escalera del entorno 5:

```
<Body name="ladder" type="static">
  <!-- Geometrías de colisión -->
  <Geom name="step0" type="box" render="false">
      <!-- Dimensiones y transformaciones -->
  </Geom>
  <Geom name="step1" type="box" render="false">
      <!-- Dimensiones y transformaciones -->
  </Geom>
  <!-- ...y más escalones... -->
  <Geom name="left_railing_long" type="cylinder" render="false">
      <!-- Dimensiones y transformaciones -->
  </Geom>
  <!-- ...y más barandillas... -->
  <!-- Geometría visual -->
  <Geom name="visual" type="sphere">
      <render>meshes/ladder.iv</render>
      <radius>0</radius>
  </Geom>
</Body>
```

Fragmento de código 3.15. Geometría visual compleja con varias geometrías de colisión en OpenRAVE

También se da este caso cuando se quiere aplicar distintas transformaciones a la geometría visual y a la geometría de colisión: es necesario utilizar etiquetas **Geom** independientes que establezcan las transformaciones que correspondan en cada caso.

- Caso 2.2. Geometría visual y de colisión mediante malla tridimensional

Queda por analizar el caso en que se quiere definir una geometría de colisión mediante una malla 3D, aunque, como ya se ha comentado, siempre se debe intentar simplificar la colisión mediante geometrías primitivas.

Para representar una geometría de colisión mediante un modelo 3D en OpenRAVE se utilizan las etiquetas **data** o **collision**, que dan la ubicación del fichero 3D. Generalmente esta etiqueta va acompañada de la etiqueta **render** para definir la geometría visual, como en el Fragmento de código 3.16.

```
<Geom type="trimesh">
    <render>meshes/o_jersey_barrier.wrl</render>
    <collision>meshes/o_jersey_barrier.wrl</collision>
</Geom>
```

Fragmento de código 3.16. Geometría de colisión compleja en OpenRAVE

- Comparación de códigos

Gazebo	OpenRAVE	
<collision name="collision"> <geometry> <!-- Geometría colisión --> </geometry> </collision> <visual name="visual"> <geometry> <!-- Geometría visual --> </geometry> </visual>	<Geom type="box"> <!-- Características visuales y de colisión --> </Geom>	Caso 1
	<Geom type="cylinder"> <!-- Geometría visual --> <render>meshes/Wheel_Front_Left.iv</render> <!-- Geometría de colisión --> <radius>0.3175</radius> <height>0.2794</height> </Geom>	Caso 2.1.1
	<!-- Geometrías de colisión --> <Geom name="step0" type="box" render="false"> <!-- Dimensiones y transformaciones --> </Geom> <!-- Geometría visual --> <Geom name="visual" type="sphere"> <render>meshes/ladder.iv</render> <radius>0</radius> </Geom>	Caso 2.1.2.
	<Geom type="trimesh"> <render>meshes/o_jersey_barrier.wrl</render> <data>meshes/o_jersey_barrier.wrl</data> </Geom>	Caso 2.2.

Tabla 3.6. Comparación de código de geometrías

Conclusiones:

- La etiqueta **geometry** en Gazebo se sustituye por **Geom** en OpenRAVE (**geometry** también está admitido en OpenRAVE).
- En Gazebo se definen siempre las geometrías visual y de colisión de forma independiente mediante las etiquetas **visual** y **collision**.
- En OpenRAVE se emplean las etiquetas **render**, **data**, **collision** o el atributo **render** (atributo de **Geom**) según las geometrías que queramos definir.
- Las etiquetas **visual** y **collision** en Gazebo admiten el atributo **name**, en OpenRAVE el atributo **name** se asigna a **Geom**; en ambos casos es opcional.

3.4.7 Articulaciones

Las articulaciones conectan dos cuerpos entre sí. Su función es limitar el movimiento relativo entre los links. Las articulaciones pueden ser de distintos tipos según las restricciones que impongan en el movimiento. En la Figura 3.4 puede verse una representación de los tipos de articulaciones más habituales: rótula (articulación esférica), bisagra (articulación cilíndrica) y deslizador (articulación prismática) [46].

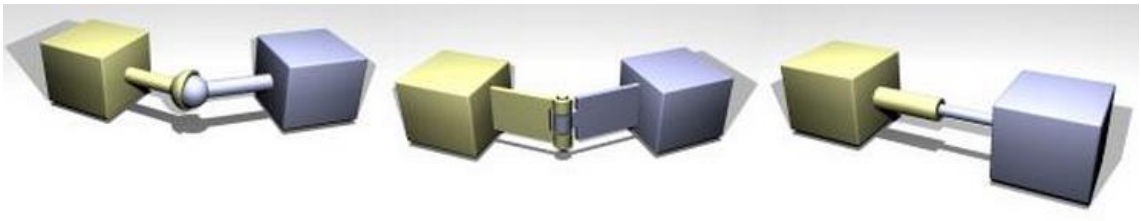

Figura 3.4. Tres tipos de articulaciones: rótula, bisagra y deslizador

En la definición de una articulación hay que identificar: los dos links que forman parte de la articulación, el tipo de articulación y las características concretas asociadas a la articulación: límites de giro o desplazamiento impuestos, posición y dirección del eje (o ejes) de giro o desplazamiento cuando proceda, límites de velocidad y torsión soportados por la articulación, etc.

Se va a analizar cómo se definen estos parámetros en Gazebo y OpenRAVE.

- Gazebo

Una articulación en Gazebo se identifica con la etiqueta **joint**. Esta etiqueta toma dos atributos: **name**, que debe ser un nombre único, y **type**, que indica el tipo de articulación de que se trata. Los posibles valores de **type** son: **revolute** (bisagra), **revolute2** (equivale a dos bisagras en serie), **prismatic** (deslizador), **ball** (rótula o esférico), **universal** (esférico con un grado de libertad restringido) y **piston** (deslizador que puede girar alrededor del eje) [47]. El tipo de articulación determinará las características que es necesario definir.

En la unión, hay un cuerpo que actúa como padre y otro que tiene la función de hijo, que se designan mediante las etiquetas **parent** y **child**, respectivamente. La posición del eje de giro o desplazamiento se define respecto al cuerpo hijo. Esto último es cierto para la versión 1.5 del formato SDF. En versiones anteriores, el eje se referencia respecto al link padre. Para permitir la adaptación de un formato a otro se crea la etiqueta **use_parent_model_frame**. Esta etiqueta, cuando tiene el valor **true**, indica que el giro o desplazamiento es relativo al link padre [48]. Mediante el uso de dicha etiqueta se pueden adaptar articulaciones creadas en la versión 1.4 de SDF a la versión 1.5 manteniendo todo el código y simplemente añadiendo una línea.

Otras características que se definen para las articulaciones son:

- posición del punto respecto al cual se realiza el movimiento, tomando el link hijo como referencia, con la etiqueta **pose**;
- eje de rotación o traslación (o ejes, en articulaciones de tipo revolute2 y universal) mediante la etiqueta **axis** y **axis2**;
- límites de giro o desplazamiento, se indican dentro de la etiqueta **axis**, con la etiqueta **limit** y las etiquetas **lower** y **upper** para definir cada límite (en radianes o metros según corresponda al tipo de articulación);
- valores máximos de velocidad y momento soportados por la articulación, definidos dentro de **limit**, como **velocity** y **effort**.

En el Fragmento de código 3.17 se puede ver la definición de una articulación de tipo bisagra, con eje de giro vertical y giro limitado entre -44.27° y 44.27°.

```
<joint name="front_left_steering_joint" type="revolute">
  <child>front_left_wheel_steering_block</child>
  <parent>chassis</parent>
  <axis>
    <xyz>0 0 1</xyz>
    <limit>
      <lower>-0.7727</lower>
      <upper>0.7727</upper>
    </limit>
  </axis>
</joint>
```

Fragmento de código 3.17. Creación de articulación en Gazebo

- OpenRAVE

OpenRAVE utiliza la etiqueta **Joint** para designar una articulación. Mediante el atributo **name** se nombra la articulación y mediante **type** se indica el tipo, siendo los tipos disponibles: **hinge**, **slider**, **universal**, **hinge2** y **spherical**, que corresponden respectivamente a articulaciones cilíndrica (bisagra), prismática, universal, de doble deslizamiento y esférica [36].

En OpenRAVE no se diferencia entre link padre y link hijo, ambos se indican dentro de etiquetas **Body** (una para cada cuerpo).

Las características específicas de la articulación se definen mediante las etiquetas:

- **anchor**, para definir la posición del punto de giro;
- **offsetfrom**, que indica el cuerpo respecto al que se realiza el giro, que correspondería al link hijo en Gazebo;
- **axis**, para articulaciones con un eje de giro o desplazamiento, o **axis1** y **axis2**, en articulaciones con dos ejes (doble deslizamiento y universal);
- **limits**, **limitsdeg** o **limitsrad**, que indican los límites al movimiento relativo de los cuerpos en metros, grados o radianes, los dos primeros valores corresponden al primer eje, los dos siguientes, al segundo, si lo hay;
- **maxvel**, **maxaccel** y **maxtorque**, dan los valores máximos de velocidad, aceleración y momento de la articulación.

Para definir la misma articulación cilíndrica que se ha visto como ejemplo en Gazebo se utiliza el Fragmento de código 3.18 en OpenRAVE.

```
<Joint name="front_left_steering_Joint" type="hinge">
  <Body>chassis</Body>
  <Body>front_left_wheel_steering_block</Body>
  <offsetfrom>front_left_wheel_steering_block</offsetfrom>
  <axis>0 0 1</axis>
  <limitsrad>-0.7727 0.7727</limitsrad>
</Joint>
```

Fragmento de código 3.18. Creación de articulación en OpenRAVE

- Correspondencia entre etiquetas y atributos

En la Tabla 3.7 se resume la jerarquía y equivalencia entre las etiquetas y atributos usados en cada simulador para describir articulaciones, según lo descrito anteriormente.

Gazebo	**OpenRAVE**
joint	Joint
atributos:	**atributos:**
name	name
type **revolute** **revolute2** **prismatic** **ball** **universal**	type **hinge** **hinge2** **slider** **spherical** **universal**
elementos:	**elementos:**
parent	Body
child	Body offsetfrom
pose	anchor
axis, axis2	
xyz	axis, axis1, axis2
limit lower upper	limits, limitsdeg, limitsrad
velocity effort	maxvel maxtorque

Tabla 3.7. Correspondencia entre etiquetas y atributos de articulaciones

Conclusiones:

- La etiqueta **joint** en Gazebo se mantiene en OpenRAVE.
- La correspondencia entre las etiquetas y atributos que definen las articulaciones en Gazebo y OpenRAVE viene reflejada en la Tabla 3.7.

3.5 CARACTERÍSTICAS GEOMÉTRICAS DE LOS ELEMENTOS

En este apartado se van a estudiar algunas características de los elementos de un entorno, concretamente aquellas relacionadas con su posición en el entorno y sus propiedades geométricas. Se analizan los sistemas de coordenadas y unidades empleados en cada simulador, la forma geométrica posible de un cuerpo, y las transformaciones aplicables a los elementos.

3.5.1 Sistema de Coordenadas

Es necesario conocer en primer lugar el sistema de coordenadas en que se apoyan los simuladores para poder definir las dimensiones de los elementos y las transformaciones aplicadas.

- Gazebo

Utiliza un sistema de coordenadas orientado a la derecha, con los ejes x e y en el plano y el eje z creciente con la altura [23].

- OpenRAVE

El sistema de coordenadas en OpenRAVE se define por giro a derechas, con ejes x e y en el plano horizontal, y eje z positivo hacia arriba, en sentido opuesto a la gravedad.

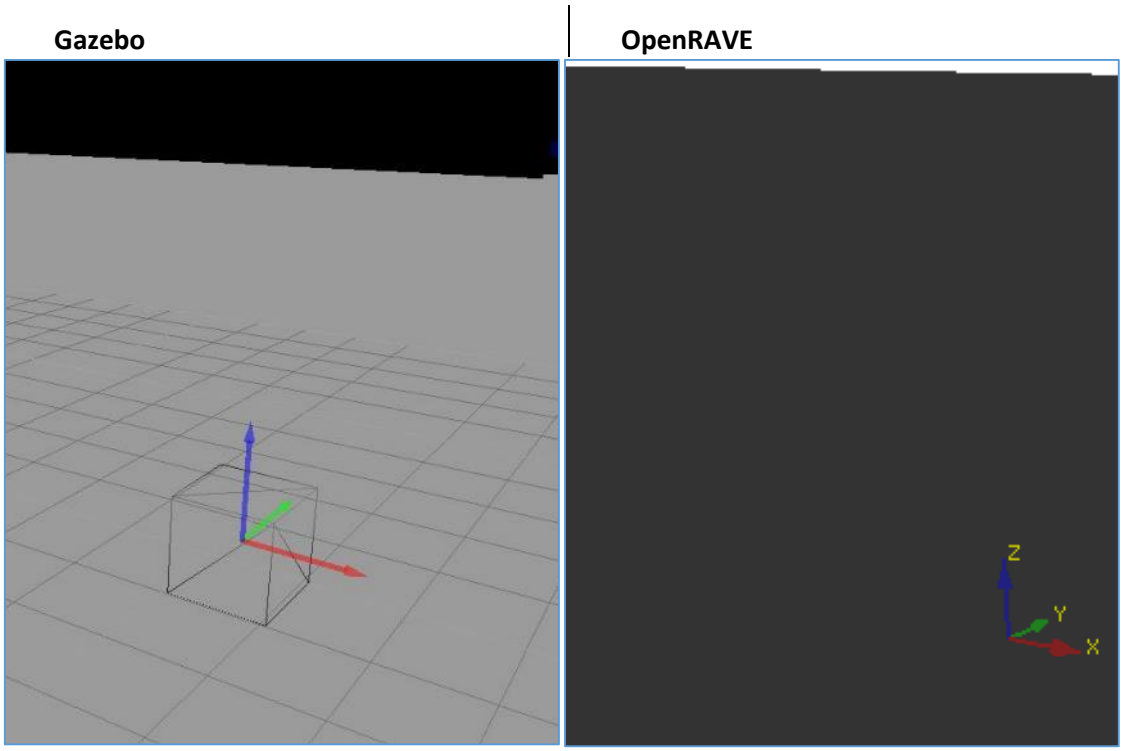

Figura 3.5. Sistema de coordenadas empleado en Gazebo (a la izquierda) y OpenRAVE (a la derecha). El eje rojo representa el eje 'x', el verde el eje'y' y el azul el eje 'z'

Conclusiones:

- Ambos utilizan el mismo sistema de coordenadas por lo que no es necesaria ninguna conversión en este sentido.

3.5.2 Unidades

Se habla de las unidades que se emplean por convención, recomendadas para facilitar la integración con otros entornos y elementos.

- Gazebo

Gazebo sigue el estándar definido para ROS ("Standard Units of measure and Coordinate Conventions"), por lo que las unidades empleadas en Gazebo son las del Sistema Internacional: metros, kilogramos y segundos [49]. Para los ángulos se emplean radianes.

- OpenRAVE

En OpenRAVE se recomienda el uso de las unidades del Sistema Internacional [50]. Los ángulos se miden por norma general en grados.

Conclusiones:

- Hay que estudiar la necesidad de adaptación de unidades para cada caso individual.

3.5.3 Formas Geométricas

Un cuerpo está formado por geometrías, visuales o de colisión, que tendrán una forma geométrica determinada. Las formas geométricas primitivas en ambos simuladores son: caja (ortoedro), cilindro, esfera y plano. Para la definición de geometrías complejas se emplean mallas de triángulos, como se ve en el apartado de Mallas Tridimensionales.

- Gazebo

La forma geométrica de un cuerpo se define dentro de la etiqueta **geometry**. Se establece el tipo de figura geométrica mediante una etiqueta. Las etiquetas posibles son: **box**, **cylinder**, **sphere**, **plane** (geometrías primitivas de Gazebo) y **mesh** (que identifica una geometría mediante malla 3D).

- OpenRAVE

La forma geométrica se define mediante el atributo **type** de la etiqueta **Geom**. Los posibles valores de **type** son **box**, **cylinder**, **sphere**, **plane** (primitivas) y **trimesh** (malla 3D).

- Definición de formas geométricas

A continuación se estudia cómo se define cada forma geométrica posible, incluyendo sus dimensiones y ejes locales, para cada programa.

- Caja

Gazebo. Se define mediante la etiqueta **box**. Las dimensiones se dan dentro de la etiqueta **size** y corresponden a las longitudes en metros de los lados x, y, z según los ejes locales del objeto.

OpenRAVE. Se indica con el atributo **type="box"**, y sus dimensiones vienen dadas por la etiqueta **extents** como las semi-longitudes de las aristas de la caja, en metros.

Comparación de códigos

Gazebo	**OpenRAVE**

`<geometry>`
`  <box>`
`  <size>6 2.5 1</size>`
`  </box>`
`</geometry>`
	`<Geom type="box">`
	`  <extents>3 1.25 0.5</extents>`
	`</Geom>`

Tabla 3.8. Comparación de código de geometría de caja

Conclusiones:

- La etiqueta **box** en Gazebo se sustituye por un atributo en OpenRAVE.
- La etiqueta **size** en Gazebo se sustituye por **extents** en OpenRAVE y los valores en OpenRAVE han de ser la mitad que en Gazebo.

- Cilindro

Gazebo. Las dimensiones de un cilindro (etiqueta **cylinder**) en Gazebo se dan como **length** y **radius**, y su orientación antes de aplicar trasformaciones es con el eje de revolución en la dirección del eje z.

OpenRAVE. Una geometría de tipo **cylinder** se define mediante las etiquetas **height** y **radius** y su eje de revolución se orienta inicialmente en la dirección del eje y.

Comparación de códigos

Gazebo	OpenRAVE

`<geometry>`
`  <cylinder>`
`  <radius>0.02413</radius>`
`  <length>1.0668</length>`
`  </cylinder>`
`</geometry>`
	`<Geom type="cylinder">`
	`  <radius>0.02413</radius>`
	`  <height>1.0668</height>`
	`</Geom>`

Tabla 3.9. Comparación de código de geometría cilíndrica

Conclusiones:

- La etiqueta **cylinder** en Gazebo se sustituye por un atributo en OpenRAVE.
- La etiqueta **radius** se emplea igual en ambos.
- La etiqueta **length** en Gazebo indica la longitud en la dirección z del cilindro y se sustituye en OpenRAVE por la etiqueta **height**, que se refiere a la dirección y.

- Esfera

Gazebo. Un elemento **sphere** se define mediante el valor dado por la etiqueta **radius**.

OpenRAVE. Tipo **sphere** y dimensión dada por el radio mediante la etiqueta **radius**.

Comparación de códigos

Gazebo	OpenRAVE

`<geometry>`
`  <sphere>`
`  <radius>1.5</radius>`
`  </cylinder>`
`</geometry>`
	`<Geom type="sphere">`
	`  <radius>1.5</radius>`
	`</Geom>`

Tabla 3.10. Comparación de código de geometría esférica

Conclusiones:

- La etiqueta **sphere** en Gazebo se sustituye por un atributo en OpenRAVE.
- La etiqueta **radius** se emplea con el mismo significado en ambos.

- Plano

Gazebo. Un plano (**plane**) en Gazebo está defino por su vector normal (cuyas coordenadas se indican en la etiqueta **normal**) y sus dimensiones en el plano perpendicular a este vector (dadas en la etiqueta **size**). Se trata, por tanto, de un plano de dimensiones finitas (no es estrictamente un plano geométrico).

OpenRAVE. Describe un plano (etiqueta **plane**) a través de su ecuación general. Dentro de la etiqueta **eqn** se definen los valores de los parámetros a, b, c y d, que definen el plano de ecuación ax+by+cz+d=0. El plano representado utilizando esta etiqueta tiene las características de un plano infinito. Un plano en OpenRAVE no admite traslaciones ni rotaciones, la posición viene fijada por su ecuación.

Comparación de códigos

Gazebo	OpenRAVE
<geometry> <plane> <normal>0 0 1</normal> <size>300 50</size> </plane> </geometry>	<Geom type="plane"> <eqn>0 0 1 0</eqn> </Geom>

Tabla 3.11. Comparación de código de geometría de plano

Conclusiones:

- La etiqueta **plane** en Gazebo se sustituye por un atributo en OpenRAVE.
- El plano se define mediante las etiquetas **normal** y **size** en Gazebo (plano finito) y mediante su ecuación con la etiqueta **eqn** en OpenRAVE (plano infinito).

- Suelo

Es un caso especial de un tipo de geometría. Se hace una mención especial a este elemento ya que se suele ser común a todos los entornos y tiene interés el estudio de su definición en cada programa.

Gazebo. En Gazebo el suelo se representa mediante una geometría de tipo **plane**. Sin embargo, como ya se ha visto, los planos en Gazebo no son infinitos, por lo que en la práctica el suelo se asemeja a una geometría de caja con una de sus dimensiones despreciable.

OpenRAVE. Debido a las características de la geometría tipo **plane** en OpenRAVE (plano infinito), es más recomendable usar una geometría de caja (**box**) con dimensión vertical despreciable para representar el suelo. Hay que tener en cuenta que habrá que aplicar una traslación correspondiente a la dimensión vertical para situar el suelo en origen (esto se hace mediante la etiqueta **translation**, que se estudia en el apartado de Transformaciones: Traslación y Rotación).

Comparación de códigos

Gazebo	OpenRAVE
<geometry> <plane> <normal>0 0 1</normal> <size>300 50</size> </plane> </geometry>	<Geom type="box"> <extents>150 25 0.001</extents> <translation>0 0 -0.001</translation> </Geom>

Tabla 3.12. Comparación de código de suelo

Conclusiones:

- El suelo en Gazebo se define mediante una geometría de tipo **plane**, en OpenRAVE, mediante una geometría tipo **box**.

- Malla

Gazebo. Se utiliza la etiqueta **mesh**.

OpenRAVE. El atributo **type** toma el valor **trimesh**.

Se referencia el modelo externo como se vio en el apartado de Mallas Tridimensionales.

Comparación de códigos

Gazebo	OpenRAVE

`<geometry>`
`  <mesh>`
`  <uri>model://meshes/block.dae</uri>`
`  </mesh>`
`</geometry>`
	`<Geom name="visual" type="trimesh">`
	`  <render>meshes/block.iv</render>`
	`</Geom>`

Tabla 3.13. Comparación de código de geometría de malla

Conclusiones:

- La etiqueta **mesh** en Gazebo equivale a dar al atributo **type** el valor **trimesh** en OpenRAVE.

3.5.4 Transformaciones: Traslación y Rotación

Las transformaciones se aplican a modelos para especificar su situación en el entorno y a cuerpos y geometrías para definir su posición concreta dentro del modelo.

- Gazebo

En Gazebo la transformación aplicada a los cuerpos se define mediante la etiqueta **pose**. En esta etiqueta se dan seis valores numéricos: los tres primeros corresponden a la traslación en metros en los ejes x, y, z; los tres últimos determinan la rotación en ángulos de Euler (yaw, pitch, roll) dados en radianes, que son respectivamente los ángulos de rotación alrededor de los ejes x, y, z globales siguiendo la regla de la mano derecha.

- OpenRAVE

En OpenRAVE la traslación y rotación se expresan en etiquetas independientes. La transformación total de un elemento es una composición de todas las traslaciones y rotaciones que se indiquen.

La etiqueta **Translation** toma tres valores, correspondientes a la traslación a lo largo de los ejes coordenados x, y, z, en metros.

La rotación de un elemento en OpenRAVE se puede aplicar de diferentes formas:

- Como rotación ángulo-eje, mediante la etiqueta **RotationAxis**. Dentro de esta etiqueta, se definen cuatro valores, que corresponden, los tres primeros, a las coordenadas del vector que nos da la dirección del eje de giro, y el cuarto, al ángulo que gira el cuerpo alrededor del eje definido, en grados.
- Mediante un cuaternión utilizando la etiqueta **quat**, cuyas cuatro coordenadas se obtienen operando con las coordenadas del eje de giro y el valor de ángulo de rotación.
- A través de la matriz asociada a la rotación, definida mediante sus nueve elementos en la etiqueta **RotationMat**.

- Comparación de códigos

Gazebo	OpenRAVE
`<pose>2.45 3.34 0 0 0 -1.5707</pose>`	`<Translation>2.45 3.34 0</Translation>` `<RotationAxis>0 0 1 -90</RotationAxis>`

Tabla 3.14. Comparación de código de transformaciones: traslación y rotación

Conclusiones:

- La etiqueta **pose** en Gazebo se sustituye por **Translation** y una etiqueta de rotación: **RotationAxis**, **quat** o **RotationMat** en OpenRAVE.
- Los valores de traslación son en metros en ambos casos.
- Los ángulos se dan en **radianes** en Gazebo y en **grados** en OpenRAVE.
- La conversión de rotaciones en Gazebo expresadas en ángulos de Euler a rotación en OpenRAVE de la forma ángulo-eje es sencilla si definimos esta última como composición de rotaciones alrededor de los ejes coordenados.

3.5.5 Transformaciones: Escalado de modelos

Cuando se importa un modelo 3D puede ser necesario añadir un factor de escalado.

- Gazebo

En Gazebo se utiliza la etiqueta **scale** para definir la escala a aplicar en el modelo, el cual ha de estar referenciado con la etiqueta **uri**. Mediante la etiqueta **scale** se define el valor del escalado para cada uno de los tres ejes coordenados.

- OpenRAVE

OpenRAVE permite el escalado de modelos incluyendo, tras la referencia al archivo 3D, dentro de la misma etiqueta, el valor de la escala a aplicar. Admite un valor único que será aplicado de forma homogénea en los tres ejes.

- Comparación de códigos

Gazebo	OpenRAVE
`<uri>model://meshes/polaris.dae</uri>` `<scale>1.003 0.886 0.886</scale>`	`<render>meshes/bar.wrl 0.01</render>`

Tabla 3.15. Comparación de código de transformaciones: escalado

Conclusiones:

- En Gazebo se aplica escalado utilizando la etiqueta **scale;** en OpenRAVE, tras el nombre del modelo a escalar.
- Gazebo define un factor de escala para cada eje, por lo que se pueden dar valores distintos para cada uno de ellos; en OpenRAVE, el escalado es homogéneo para los tres ejes.

3.6 OTRAS CARACTERÍSTICAS DEL ENTORNO Y SUS ELEMENTOS

3.6.1 Materiales y Color

- Gazebo

Gazebo permite incorporar texturas a sus modelos. El material empleado en un modelo se define dentro de un elemento **visual**, mediante la etiqueta **material**. Los materiales disponibles en Gazebo están definidos en un fichero independiente, y se referencia el material concreto a emplear mediante la etiqueta **script**. Los colores sólidos en Gazebo también se definen mediante la etiqueta **material**.

- OpenRAVE

El lenguaje XML propio de OpenRAVE no admite la definición de materiales asociados a sus elementos. Por tanto, en OpenRAVE sólo es posible representar texturas cuando se cargan

mallas tridimensionales. Como alternativa, existe la posibilidad de definir un color sólido para el modelo que se aproxime a la textura que se quiere representar. Para ello se utiliza la etiqueta **diffuseColor** dentro de la etiqueta **Geom**. Esta etiqueta toma tres valores correspondientes al código RGB del color.

- Comparación de códigos

Texturas

Gazebo	OpenRAVE

`<material>`
`  <script>`
`  <name>Gazebo/WoodFloor</name>`
`  </script>`
`</material>`
	`<diffuseColor>0.4 0.2 0</diffuseColor>`

Tabla 3.16. Comparación de código de texturas

Colores

Gazebo	OpenRAVE
`<material>Gazebo/Orange</material>`	`<diffuseColor>1 0.4 0.04</diffuseColor>`

Tabla 3.17. Comparación de código para la definición de colores

Conclusiones:

- En Gazebo se definen texturas y colores mediante la etiqueta **material**. En OpenRAVE no es posible definir texturas. Los colores en OpenRAVE se incluyen con la etiqueta **diffuseColor**.

3.6.2 Cámara

Es posible definir en el archivo de entorno la posición inicial de la cámara, es decir, especificar el ángulo y la posición iniciales desde los que se va a ver el entorno.

- Gazebo

La cámara en Gazebo se incluye dentro de la etiqueta **gui**, que define características de la visualización del entorno. Mediante la etiqueta **camera** se especifica el nombre (etiqueta **name**) y posición (etiqueta **pose**) de la cámara en el entorno.

- OpenRAVE

Para posicionar la cámara en el entorno OpenRAVE se vale de las etiquetas **camtrans**, **camrotationaxis** y **camfocal**, que proporcionan la información relativa a la situación de la cámara en el espacio.

- Comparación de códigos

Gazebo	OpenRAVE

`<gui>`
`  <camera>`
`  <name>camara1</name>`
`  <pose>8 -3 3 128 0 0</pose>`
`  </camera>`
`</gui>`
	`<camtrans>8 -3 3</camtrans>`
	`<camrotationaxis>1 0 0 128</camrotationaxis>`
	`<camfocal>37.935791</camfocal>`

Tabla 3.18. Comparación de código de posición de la cámara

Conclusiones:

- La cámara se sitúa en Gazebo utilizando la etiqueta **pose** dentro de **camera**. En OpenRAVE, se emplean las etiquetas **camtrans**, **camrotationaxis** y **camfocal** para definir los distintos parámetros de la cámara.
- En Gazebo es obligatorio especificar un nombre para la cámara, mediante la etiqueta **name**. En OpenRAVE no se nombra la cámara.

3.6.3 Sensores

Un sensor permite recoger información sobre el entorno. Los datos obtenidos pueden ser de naturaleza visual, sonora, táctil, etc. El sensor puede estar asociado a un modelo o formar parte directamente del entorno.

- Gazebo

Mediante la etiqueta **sensor** se incluye un sensor en el entorno. El atributo **type** define el tipo de sensor de que se trata. Los posibles tipos son: **camera**, **contact**, **gps**, **imu**, **ray**, **sonar**, **transceiver** y **forcetorque**. Las propiedades específicas del sensor se describen dentro de una etiqueta del mismo nombre que el tipo. El Fragmento de código 3.19 es un ejemplo de la definición de un sensor de contacto.

```
<sensor name="seat_contact" type="contact">
  <always_on>true</always_on>
  <update_rate>1000</update_rate>
  <contact>
    <collision>mud_seat</collision>
  </contact>
</sensor>
```

Fragmento de código 3.19. Definición de sensor de contacto en Gazebo

- OpenRAVE

Un sensor en OpenRAVE se puede definir como parte de un robot o como un elemento del entorno. En el primer caso se describen dentro de la etiqueta **AttachedSensor**, en el segundo se emplea la etiqueta **sensor**. El tipo de sensor se indica mediante el atributo **type**. OpenRAVE soporta los siguientes tipos de sensores: **laser**, **camera**, **JointEncoder**, **Force6D**, **IMU**, **Odometry**, **Tactile**, **Actuator** [51]. Sin embargo, no viene con la implementación de todos ellos, por lo que sólo es posible acceder a algunos directamente. Los sensores que vienen incluidos en OpenRAVE son: *BaseCamera* (tipo cámara o visual), *BaseLaser* y *FlashLidar* (ambos de tipo láser) [52]. Para poder utilizar otros sensores es necesario emplear plugins que definan el sensor con el que se pretenda trabajar. Para esto se puede utilizar la herramienta de simulación OpenGrasph[12] [53]. En el Fragmento de código 3.20 se presenta la definición de un sensor de tipo visual que se emplea en un robot.

[12] OpenGRASP es una herramienta de simulación que proporciona plugins para OpenRAVE que permiten ampliar las funcionalidades de este programa. Se centra fundamentalmente en el desarrollo de habilidades de destreza manual de los robots [75].

```
<Robot>
  <AttachedSensor>
    <sensor type="BaseCamera" args="">
      <KK>640 480 320 240</KK>
      <width>640</width>
      <height>480</height>
      <framerate>5</framerate>
    </sensor>
  </AttachedSensor>
</Robot>
```

Fragmento de código 3.20. Definición de un sensor visual en OpenRAVE

Conclusiones:

- Los sensores se definen con la etiqueta **sensor** tanto en Gazebo como en OpenRAVE. La posibilidad de adaptación de Gazebo a OpenRAVE dependerá de que el tipo de sensor empleado en Gazebo sea uno de los que están ya creados para OpenRAVE.

3.7 MOTORES FÍSICOS

Un motor físico es una herramienta que permite simular de forma aproximada sistemas físicos tales como la dinámica del sólido rígido, incluyendo detección de colisiones. Para programas de simulación se emplean motores físicos capaces de responder en tiempo real, que utilizan cálculos simplificados para mejorar la velocidad de procesado [54]. Este tipo de motores físicos son los que se emplean en Gazebo y OpenRAVE.

- Gazebo

Los motores físicos disponibles en Gazebo son los siguientes: Open Dynamics Engine (ODE), Bullet, Simbody, y DART [16]. Los tres primeros vienen con el paquete de Gazebo, para usar DART es necesario compilar desde fuentes.

Se puede escoger fácilmente el motor físico a emplear en una simulación, mediante una instrucción en la ventana de comandos o indicándolo en el archivo de entorno, como se verá en el apartado de Propiedades físicas del entorno [55].

- OpenRAVE

OpenRAVE tiene distintos motores físicos disponibles: Bullet (la librería integrada en OpenRAVE se denomina bulletrave), ODE (en OpenRAVE se utiliza la correspondiente librería derivada: oderave) y un motor físico interno definido para OpenRAVE (GenericPhysicsEngine) [56]. Bullet no viene integrada en el código de OpenRAVE, para poder utilizarla previamente hay que compilar utilizando las fuentes [56].

Para el cálculo de colisiones, las herramientas disponibles son ODE, Bullet (necesario compilar desde fuentes), un detector de colisiones interno (GenericCollisionChecker) y PQP[13] (este último se emplea principalmente para el cálculo de distancias entre cuerpos) [56].

3.7.1 ODE

Entre los motores disponibles en cada simulador, ODE es el que se emplea en la definición de las pruebas de DRC en Gazebo. Es por esto que en la realización de este proyecto se va a

[13] PQP (Proximity Query Package) es una librería que permite detectar colisiones y calcular distancias entre cuerpos geométricos [71].

trabajar con ODE también en OpenRAVE. A continuación se van a estudiar las características de esta librería.

ODE (Open Dynamics Engine) es un proyecto de Russell Smith, creador y principal desarrollador de esta librería de código abierto. ODE se emplea para la simulación de la dinámica del sólido rígido, especialmente para cuerpos articulados: vehículos terrestres, criaturas andadoras o grupos de objetos. Se trata de una plataforma multifunción, estable y madura, a la vez que rápida y flexible, que permite la simulación de objetos en movimiento en entornos de realidad virtual cambiantes. Soporta articulaciones complejas e integra detección de colisiones y fricción [57], [46]. Algunos de los proyectos que usan ODE son: videojuegos, plataformas de realidad virtual y, por supuesto, programas de simulación de robots, entre ellos Gazebo y OpenRAVE [58].

Un cuerpo rígido tiene asociadas una serie de propiedades dinámicas, que son las que se van a utilizar en las simulaciones. Algunas propiedades varían o pueden variar en el tiempo: la posición y orientación de los cuerpos, así como su velocidad linear y angular; mientras que otras propiedades se mantienen constantes a lo largo de la simulación: la masa, la posición del centro de masas o centro de gravedad y la matriz de inercia, que describe cómo está distribuida la masa del sólido respecto a su centro de gravedad. La forma y dimensiones del cuerpo también son constantes, pero no se trata de propiedades dinámicas, únicamente se utilizan en la simulación para la detección de colisiones [59]. ODE tiene además la funcionalidad necesaria para detectar colisiones, pero permite el uso de otras herramientas para este fin. Es decir, se puede usar ODE como motor físico y emplear una herramienta distinta para la detección de colisiones.

3.7.2 Propiedades dinámicas de los cuerpos

Las propiedades dinámicas que se definen para un cuerpo son masa, posición del centro de gravedad y matriz de inercia.

- Gazebo

Las propiedades dinámicas o propiedades de inercia en Gazebo se indican mediante la etiqueta **inertial**. Dentro de esta etiqueta se incluyen las etiquetas **mass**, **pose** (u **origin**) e **inertia**, para definir, respectivamente, las propiedades de masa, posición del centro de gravedad y matriz de inercia.

La etiqueta **inertia** representa el tensor de inercia. Como se trata de una matriz simétrica, sólo son necesarios 6 valores, no se especifican los elementos dependientes. Dentro del tensor, se utilizan las etiquetas **ixx**, **ixy**, **iyy**, etc., para indicar el valor de cada elemento de la matriz.

- OpenRAVE

Para definir las propiedades asociadas a la masa en OpenRAVE, se utiliza la etiqueta **mass**. Esta etiqueta tiene un atributo, **type**, que define el tipo de masa. El tipo de masa puede ser: **box**, **sphere**, **mimicgeom** y **custom**. Los tres primeros van a calcular la posición del centro de gravedad y matriz de inercia automáticamente según los valores de masa o densidad. En el caso de **custom** hay que definir estos valores manualmente. No se puede asociar el tipo **custom** a geometrías definidas por mallas 3D o planos.

Las etiquetas **total**, **density**, **com**, **inertia** se emplean dentro de la etiqueta **mass** para indicar, respectivamente, valor de la masa en kilogramos, densidad, posición del centro de gravedad y matriz de inercia (definida por 9 valores, incluidas las coordenadas dependientes).

- Comparación de códigos

Gazebo	OpenRAVE

`<inertial>`
`  <pose>0 0 0.0714375 0 0 0</pose>`
`  <mass>5</mass>`
`  <inertia>`
`  <ixx>0.06908</ixx>`
`  <ixy>0.0000</ixy>`
`  <iyy>0.028705</iyy>`
`  <ixz>0.0000</ixz>`
`  <iyz>0.0000</iyz>`
`  <izz>0.073708</izz>`
`  </inertia>`
`</inertial>`
	`<mass type="custom">`
	`  <com>0 0 .0714375</com>`
	`  <total>5</total>`
	`  <inertia>0.06908 0 0 0 0.028705 0 0 0 0.073708</inertia>`
	`</mass>`

Tabla 3.19. Comparación de código de propiedades dinámicas de los cuerpos

Conclusiones:

- Las propiedades dinámicas se describen en Gazebo dentro de la etiqueta **inertial** y en OpenRAVE en la etiqueta **mass**.
- La etiqueta **pose** en Gazebo se sustituye por **com** en OpenRAVE. En esta etiqueta se está dando la posición de un punto, por lo que las rotaciones indicadas en la etiqueta pose, si las hay, se ignoran.
- La masa del cuerpo se define con la etiqueta **mass** en Gazebo y **total** en OpenRAVE, en ambos se da en kilogramos.
- Mediante la etiqueta **inertia** se dan los elementos de la matriz de inercia asociada al cuerpo. En Gazebo se dan los seis elementos independientes, mientras que en OpenRAVE es necesario indicar los nueve valores.

3.7.3 Propiedades físicas del entorno

Para trabajar en entornos lo más reales posible es necesario considerar la gravedad que afecta al entorno. Se incluyen también en la simulación parámetros que aproximan el comportamiento de elementos en un entorno real. La fricción puede definirse como una propiedad global del entorno o específica de un elemento.

- Gazebo

La física del entorno se define en Gazebo dentro de la etiqueta **physics**. Mediante el atributo **type** de la misma se determina el motor físico a emplear. Las variables físicas que pueden definirse en Gazebo dentro de la etiqueta **physics** son, entre otras: **gravity**, **cfm**, **erp**[14] [60].

Gazebo define la fricción como una característica asociada a un elemento individual. En el Fragmento de código 3.21 se ve un ejemplo del uso de las etiquetas **mu** y **mu2** para definir la fricción de un elemento en las dos direcciones principales. La fricción se asocia a un elemento de colisión.

[14] cfm (constraint force mixing) y erp (error reduction parameter) son dos parámetros que permiten controlar características de la simulación, relacionadas con el contacto entre elementos [57].

```
<surface>
  <friction>
    <ode>
      <mu>100</mu>
      <mu2>50</mu2>
    </ode>
  </friction>
</surface>
```

Fragmento de código 3.21. Definición de fricción en un elemento de colisión

- OpenRAVE

Para definir el motor físico que vamos a utilizar en OpenRAVE éste se indica mediante la etiqueta **physicsengine**. Dentro de esta etiqueta incluimos **odeproperties**, que nos permite definir las propiedades físicas de nuestro entorno utilizando la librería ODE. Algunas de las propiedades que pueden definirse son: **friction**, **gravity**, **erp**, **cfm** [61].

La etiqueta **physicsengine** se emplea como hija de la etiqueta **environment** y sólo permite definir propiedades globales del entorno, no se pueden especificar características físicas relativas a un elemento individual.

- Comparación de códigos

Gazebo	**OpenRAVE**

`<physics type="ode">`
`  <gravity>0 0 -9.81</gravity>`
`  <ode>`
`  <constraints>`
`  <cfm>0.0</cfm>`
`  <erp>0.2</erp>`
`  </constraints>`
`  </ode>`
`</physics>`
	`<physicsengine type="ode">`
	`  <odeproperties>`
	`  <gravity>0 0 -9.81</gravity>`
	`  <cfm>0.0</cfm>`
	`  <erp>0.2</erp>`
	`  <friction>100</friction>`
	`  </odeproperties>`
	`</physicsengine>`

Tabla 3.20. Comparación de código de propiedades físicas del entorno

Conclusiones:

- Las propiedades físicas del entorno se describen en Gazebo dentro de la etiqueta **physics** y en OpenRAVE en la etiqueta **physicsengine**.
- La gravedad se define mediante la etiqueta **gravity**, de igual forma en ambos programas.
- Las etiquetas **cfm** y **erp** indican los valores que reciben estos parámetros.
- La fricción es una característica global del entorno en OpenRAVE y se define con la etiqueta **friction**. En Gazebo se puede especificar la fricción para cada elemento.
- En Gazebo se define la fricción en las dos direcciones principales (**mu** y **mu2**). En OpenRAVE se indica un único valor que se asigna a ambas direcciones como una fricción global.

3.7.4 Dinámico vs. Estático

Un elemento estático es aquel que es ignorado por el motor físico. Un elemento dinámico, es por tanto, aquel que está afectado por la física del entorno, como la gravedad.

- Gazebo

Se define un elemento como estático o dinámico utilizando la etiqueta **static**. La etiqueta **static** se aplica a un modelo, y puede tomar los valores **true** (indica que se trata de un modelo estático) o **false** (es el valor por defecto, e indica que el modelo ha de tratarse como dinámico).

- OpenRAVE

En OpenRAVE la característica de estático o dinámico se define como un atributo de un cuerpo (link). Un link en OpenRAVE admite el atributo **type**, que toma los valores **static** o **dynamic**, siendo **dynamic** el valor por defecto.

Puesto que en OpenRAVE esta característica no se aplica a modelos o cuerpos cinemáticos sino que se define individualmente para cada uno de los cuerpos o links que lo componen, debe definirse cada uno de ellos como estático para que todo el modelo completo sea considerado estático.

- Comparación de códigos

Gazebo

```
<model name="wood_slats">
  <static>true</static>
  <link name="link">
      <!-- propiedades link -->
  </link>
</model>
```

OpenRAVE

```
<KinBody name="wood_slats">
  <Body name="link" type="static">
      <!-- propiedades link -->
  </Body>
</KinBody>
```

Tabla 3.21. Comparación de código de definición de elementos estáticos

Conclusiones:

- La propiedad de estatismo se aplica a un modelo en Gazebo y a un link en OpenRAVE.
- En Gazebo se emplea la etiqueta **static**, que puede tomar los valores **true** o **false**. En OpenRAVE se utiliza el atributo **type**, dándole el valor **static** o **dynamic**.
- Si no se especifica esta propiedad, el valor por defecto es generar elementos dinámicos.

3.8 RECOPILACIÓN DE RESULTADOS

En la Tabla 3.22 se resumen en forma esquemática todos los resultados que se han descrito en detalle a lo largo del capítulo.

	Descripción	Gazebo	OpenRAVE
Lenguaje	Descripción del entorno y elementos	Formato SDF	Formato XML
	Descripción de robots	SDF o URDF	XML o COLLADA
Archivos	Entorno	*.world	*.env.xml
	Modelos	model.sdf	*.kinbody.xml
	Robots	model.sdf o model.urdf	*.robot.xml o *.dae
	Mallas 3D	COLLADA STL	Open Inventor VRML

Elementos del Entorno	Entorno	<world>	<Environment>
	Modelos	<model>	<KinBody>
	Robots	<model>	<Robot> <KinBody>
	Referenciar modelos externos	<include> <uri>	file="..."
	Cuerpos o links	<link>	<Body>
	Geometrías	<visual> <geometry> <collision> <geometry>	<Geom>
	Articulaciones	<joint> <child> <parent> <axis> <xyz> <limit>	<Joint> <Body> <Body> <offsetfrom> <axis> <limitsrad>
Características Geométricas de los Elementos	Formas geométricas:	<geometry>	<Geom
	Caja	<box> <size>	type="box"> <extents>
	Cilindro	<cylinder> <radius> <length>	type="cylinder"> <radius> <height>
	Esfera	<sphere> <radius>	type="sphere"> <radius>
	Plano	<plane> <normal> <size>	type="plane"> <eqn>
	Malla 3D	<mesh> <uri>	type="trimesh"> <render> **o** <collision> **o** <data>
	Transformaciones: Traslación y Rotación	<pose>	<Translation> <RotationAxis>
	Transformaciones: Escalado	<scale>	**Dentro de la etiqueta** <render> **o** <data> **o** <collision>
Otras características	Texturas	<material>	<DiffuseColor>
	Transparencia	<transparency>	<transparency>
	Cámara	<gui> <camera> <pose>	<camtrans> <camrotationaxis <camfocal>
	Sensores	<sensor type="...">	<Attachedsensor> <sensor type="...">
Física del entorno	Propiedades dinámicas de los elementos	<inertial> <mass> <pose> <inertia>	<mass> <total> <com> <inertia>
	Propiedades físicas del entorno	<physics type="ode"> <gravity> <ode> <constraints> <cfm> <erp>	<physicsengine type="ode"> <odeproperties> <gravity> <cfm> <erp>
	Dinámico vs. Estático	<model> <static>**true**</static>	<Body type="static">

Tabla 3.22. Recopilación de Resultados

4 ADAPTACIÓN DE LOS ENTORNOS A OPENRAVE

En el Capítulo 3 se ha estudiado toda la teoría relacionada con la creación de entornos en Gazebo y OpenRAVE. Este apartado se centra en cómo aplicar la teoría para, partiendo de un entorno definido para Gazebo, ser capaces de generar la misma escena en OpenRAVE. Concretamente se va a analizar cómo se han adaptado los entornos de las pruebas de DRC (creados para Gazebo) para su uso en OpenRAVE.

Se va a hablar de la conversión de los entornos de forma general, con mención explícita a dos elementos que, por su complejidad o características, se considera que merecen especial atención en este capítulo: el vehículo que forma parte del entorno de la tarea 1, y el robot Atlas creado por Boston Dynamics.

4.1 CONVERSIÓN DE LOS ENTORNOS

A continuación se van a describir los pasos para convertir un entorno creado para Gazebo en un entorno en formato de OpenRAVE.

Primero hay que identificar los archivos de los que depende el entorno en Gazebo y determinar qué archivos son necesarios para su recreación en OpenRAVE. Es decir, si se tiene un archivo de entorno en Gazebo (**.world*), será necesario otro equivalente en OpenRAVE (**.env.xml*); o, si en el entorno de Gazebo se carga un modelo externo (definido por los ficheros *model.sdf* y *config.sdf*), habrá que crear un archivo de modelo en OpenRAVE (**.kinbody.xml*).

Una vez definido el tipo de archivo, el siguiente paso es adaptar el código en Gazebo al formato propio de OpenRAVE. Para ello se han de tener en cuenta las consideraciones sobre la definición de elementos y características en formato XML que se han visto a lo largo del Capítulo 3.

Por último, hay que comprobar si en algún modelo existen dependencias con ficheros de mallas en 3D. Si es así, estos ficheros de mallas que se emplean en Gazebo (generalmente en formato COLLADA) han de convertirse a un formato válido en OpenRAVE (formato VRML u Open Inventor).

Los dos primeros pasos se realizan de forma manual; para el tercer paso, nos vamos a apoyar en herramientas de modelado en 3D que realizan la conversión entre formatos de manera automática.

4.1.1 Creación de archivos

Ya se vio en el Capítulo 3 cuál es la correspondencia entre formatos de archivos en Gazebo y OpenRAVE. Un fichero **.world* en Gazebo se convierte en **.env.xml* en OpenRAVE; un modelo definido por los modelos *model.sdf* y *config.sdf* dentro de la carpeta correspondiente equivale a un fichero **.kinbody.xml* en OpenRAVE, y un archivo de robot (que en Gazebo puede ser **.urdf* o estar definido en una carpeta al igual que un modelo) se genera en OpenRAVE como **.robot.xml*. Esto se refleja en el siguiente esquema:

Gazebo		OpenRAVE
*.world	→	*.env.xml
carpeta_modelo ├→ model.sdf └→ config.sdf	→	*.kinbody.xml
*.urdf o carpeta	→	*.robot.xml

4.1.2 Adaptación código XML

Ya se han definido los ficheros necesarios. Lo siguiente es redactar el código.

A continuación se va a ver, para cada uno de los entornos de las tareas de DRC, cuál es el código inicial en Gazebo, y cuál sería un código equivalente en OpenRAVE. Puesto que mostrar el código completo de cada uno de los archivos resultaría en un documento considerablemente extenso y ya que gran parte del código no aporta nueva información, para este estudio se seleccionan fragmentos que dan una muestra de las diferentes estructuras que aparecen en los ficheros. El resto del código que no aparece aquí reflejado, mantiene la misma estructura y se puede convertir siguiendo las directrices que se resumen en este documento.

Para realizar el análisis del código se utiliza una configuración en columnas. En la columna de la izquierda se muestra el código que aparece en los archivos de Gazebo; a la derecha, el código que corresponde a éste en OpenRAVE. Para facilitar la comprensión, se intenta que las líneas con significados equivalentes queden a la misma altura, aunque debido a las diferencias existentes entre los formatos no siempre es posible. Se incluyen explicaciones sobre la conversión en los puntos del código en que se estima conveniente dar algún apunte.

- Tarea 1

- Archivo de entorno

Fichero inicial en Gazebo	**Fichero final en OpenRAVE**
`Nombre: drc_practice_task_1.world`	`Nombre: drc_task_1.env.xml`

Los archivos SDF se inician con la etiqueta **sdf** en la cabecera.

```
<sdf version="1.4">
```

Declaración de entorno.

```
<world name="trials_task_1">
```

```
<Environment>
```

Definición de las propiedades físicas del entorno.

```
 <physics type="ode">

  <gravity>0 0 -9.81</gravity>
  <ode>
   <constraints>
    <cfm>0.0</cfm>
    <erp>0.2</erp>
   </constraints>
```

```
 <physicsengine type="ode">
  <odeproperties>
   <gravity>0 0 -9.81</gravity>


   <cfm>0.0</cfm>
   <erp>0.2</erp>
```

La fricción se define de forma global en OpenRAVE, mientras que en Gazebo se puede asociar a elementos individuales o parejas de elementos.

```
  </ode>
 </physics>
```

```
   <friction>100</friction>
  </odeproperties>
 </physicsengine>
```

Referencia a un modelo externo.

```
  <include>
   <uri>model://drc_practice_orange_jersey_barrier</uri>
   <name>barrier_4</name>
```

```
 <KinBody name="barrier_4" file="models/task_1/drc_practice_orange_jersey_barrier.kinbody.xml">
```

Las transformaciones en Gazebo se aplican mediante una etiqueta **pose**; en OpenRAVE se definen como composición de traslación más rotación. La adaptación de la traslación es directa; para las rotaciones hay que calcular el giro alrededor de cada uno de los ejes coordenados, haciendo el cambio a radianes, e indicarla en OpenRAVE en formato ángulo-eje utilizando la etiqueta **RotationAxis**. Para definir una rotación alrededor de varios ejes se emplea una etiqueta **RotationAxis** por cada una.

```
   <pose>2.457982 3.341993 0 0 0 -1.5707</pose>
  </include>
 </world>
</sdf>
```

```
  <Translation>2.457982 3.341993 0</Translation>
  <RotationAxis>0 0 1 -90</RotationAxis>
 </KinBody>
</Environment>
```

- Archivo de modelo

Fichero inicial en Gazebo	**Fichero final en OpenRAVE**
Nombre modelo: drc_practice_orange_jersey_barrier Archivo: model.sdf	Nombre:drc_practice_orange_jersey_barrier.kinbody.xml

```
<sdf version="1.5">
```

Declaración del modelo.

```
 <model name="drc_practice_orange_jersey_barrier">
```

```
<KinBody name="drc_practice_orange_jersey_barrier">
```

La propiedad de estático se asocia a un modelo en Gazebo y a un link en OpenRAVE.

```
 <static>true</static>
```

Declaración de un cuerpo que forma parte de un modelo.

```
  <link name="link">
```

```
 <Body name="link" type="static">
```

Declaración de geometrías definidas mediante mallas 3D.

```
  <visual name="visual">
    <geometry>
     <mesh>
      <uri>model://drc_practice_orange_jersey_barrier/meshes/jersey_barrier.dae</uri>
     </mesh>
    </geometry>
   </visual>
   <collision name="collision">
    <geometry>
     <mesh>
      <uri>model://drc_practice_orange_jersey_barrier/meshes/jersey_barrier.dae</uri>
     </mesh>
    </geometry>
   </collision>
  </link>
 </model>
</sdf>
```

```
  <Geom type="trimesh">
   <render>meshes/o_jersey_barrier.wrl 0.0254</render>
   <collision>meshes/o_jersey_barrier.wrl 0.0254</collision>
  </Geom>
 </Body>
</KinBody>
```

- Archivo de modelo

Fichero inicial en Gazebo	Fichero final en OpenRAVE
Nombre modelo: drc_practice_angled_barrier_45 Archivo: model.sdf	Nombre:drc_practice_angled_barrier_45.kinbody.xml
<sdf version="1.5"> <model name="drc_practice_angled_barrier_45"> <static>true</static>	<KinBody name="drc_practice_angled_barrier_45">

Es posible incluir un modelo como parte de otro modelo. En OpenRAVE, el modelo hijo no admite el atributo **name**, en su lugar se utiliza **prefix**.

<include> <name>one</name> <uri>model://drc_practice_orange_jersey_barrier</uri> <pose>0 0 0 0 0 -1.5707</pose> </include> </model> </sdf>	<KinBody prefix="one_" file="drc_practice_orange_jersey_barrier.kinbody.xml"> <Translation>0 0 0</Translation> <RotationAxis>0 0 1 -90</RotationAxis> </KinBody> </KinBody>

- Tarea 2

- Archivo de entorno

Fichero inicial en Gazebo	Fichero final en OpenRAVE
Nombre: drc_practice_task_2.world	Nombre: drc_task_2.env.xml
<sdf version="1.4"> <world name="trials_task_2">	<Environment>

El elemento suelo es un elemento básico en todos los entornos, por ello tiene interés definirlo en un fichero independiente, lo que permite usarlo en distintos entornos sin necesidad de duplicar código.

<!-- A ground plane --> <include> <uri>model://ground_plane</uri> <pose>0 0 0 0 0 0</pose> </include>	<!-- A ground plane --> <KinBody name="Floor" file="models/ground_plane.kinbody.xml"/>

Declaración de un modelo dentro del entorno.

<model name="ramp_0"> <pose>-2 0 0 0 0 0</pose> <static>true</static> <link name="link">	<KinBody name="ramp_0"> <Translation>-2 0 0</Translation> <Body name="link" type="static">

Declaración de una geometría con la misma geometría visual y de colisión.

<collision name="incline_collision"> <pose>0.29175 0 0.0335 0 0.261799364 0</pose> <geometry> <box>	<Geom name="incline" type="box"> <Translation>0.29175 0 0.0335</Translation> <RotationAxis>0 1 0 15</RotationAxis>

Las dimensiones de un elemento caja en OpenRAVE son las semilongitudes de las aristas.

<size>.631104356 2.43839619 .1</size> </box> </geometry>	<extents>.315552178 1.219198095 .05</extents>

```
    </collision>
    <visual name="incline_visual">
     <pose>0.29175 0 0.0335 0 0.261799364
0</pose>
     <geometry>
     <box>
     <size>.631104356 2.43839619 .1</size>
     </box>
     </geometry>
```

En Gazebo se pueden incluir texturas en los modelos, en OpenRAVE no es posible, lo que se hace es asemejar el aspecto del modelo coloreándolo.

```
     <material>
      <script>
       <name>Gazebo/WoodFloor</name>
      </script>
     </material>
    </visual>
   </link>
  </model>
```

```
    <diffuseColor>0.4 0.2 0</diffuseColor>


      </Geom>
    </Body>
  </KinBody>
```

La estructura de archivos es en cierta medida flexible. Por ejemplo, en el código del entorno de Gazebo se han identificado modelos de cierta complejidad que se repetían a lo largo del código del archivo de entorno. Para simplificar el código, se ha creado un nuevo archivo de modelo independiente que se puede emplear en el código sin necesidad de definir el modelo en cada aparición.

```
  <model name="angle_double_0">


   <pose>12.843472 -0.996237 0 0 0 -
1.570796</pose>

    ...
  </model>
 </world>
</sdf>
```

```
  <KinBody name="angle_double_0"
file="models/task_2/angle_double.kinbody.x
ml">
    <Translation>12.843472 -0.996237
0</Translation>
    <RotationAxis>0 0 1 -90</RotationAxis>


  </KinBody>
</Environment>
```

Para crear un nuevo fichero de modelo que sustituya a un modelo definido en el código del entorno se siguen las mismas directrices del formato XML.

Código dentro del entorno de Gazebo

```
...
<model name="angle_double_0">
 <include>
  <name>block_130</name>
  <static>true</static>
  <uri>model://cinder_block_wide</uri>

  <pose>0 -0.035859 0.050000 0 0.261794707
1.5707</pose>

 </include>
</model>
...
```

Fichero de modelo en OpenRAVE

```
<KinBody name="angle_double">


 <KinBody prefix="block_0"
file="cinder_block_wide.kinbody.xml">

  <Translation>0 -0.035859
0.050000</Translation>
  <RotationAxis>0 1 0 15</RotationAxis>
  <RotationAxis>0 0 1 90</RotationAxis>
 </KinBody>
</KinBody>
```

- Archivo de modelo: Suelo

Fichero inicial en Gazebo

Nombre modelo: ground_plane

Archivo: model.sdf

```
<sdf version="1.5">
  <model name="ground_plane">
    <static>true</static>
    <link name="link">
      <collision name="collision">
```

Fichero final en OpenRAVE

Nombre: ground_plane.kinbody.xml

```
<KinBody name="ground_plane">

 <Body name="link" type="static">
```

El suelo se representa en Gazebo mediante una geometría de tipo plano que se puede asemejar a una geometría de caja en OpenRAVE.

```
        <geometry>
          <plane>
            <normal>0 0 1</normal>
            <size>100 100</size>
```

```
  <Geom type="box">
     <extents>50 50 0.001</extents>
```

Es necesario aplicar una traslación en OpenRAVE para situar el suelo en la posición z=0.

```
          </plane>
        </geometry>
```

```
     <translation>0 0 -0.001</translation>
```

En Gazebo se define la fricción para el suelo individualmente. En OpenRAVE esto no es posible, como aproximación, se toma la fricción del suelo como fricción global para el entorno.

```
        <surface>
          <friction>
            <ode>
```

OpenRAVE sólo considera la fricción en la dirección principal, por lo que obviamos la definición de **mu2** que se da en Gazebo.

```
              <mu>100</mu>
              <mu2>50</mu2>
            </ode>
          </friction>
        </surface>
      </collision>
      <visual name="visual">
        <geometry>
          <plane>
            <normal>0 0 1</normal>
            <size>100 100</size>
          </plane>
        </geometry>
        <material>
          <script>
            <uri>file://media/materials/scripts/gazebo.material</uri>
            <name>Gazebo/Grey</name>
          </script>
        </material>
      </visual>
    </link>
  </model>
</sdf>
```

```
     <diffuseColor>.8 .8 .8</diffuseColor>


  </Geom>
 </Body>
</KinBody>
```

- Archivo de modelo

Fichero inicial en Gazebo	**Fichero final en OpenRAVE**
Nombre modelo: cinder_block_2 Archivo: model.sdf	Nombre: cinder_block_2.kinbody.xml
`<sdf version="1.5">` `  <model name="cinder_block_2">`	`<KinBody name="cinder_block_2">`

En Gazebo el modelo se define como estático cuando se referencia en el código del entorno. En OpenRAVE tenemos que definir todos los cuerpos que componen el modelo como estáticos.

`<link name="link">`	`<Body name="link" type="static">`

Definición de las propiedades dinámicas del cuerpo. Se especifican la posición del centro de gravedad y la matriz de inercia de forma manual, por lo que se declaran las propiedades como tipo **custom** en OpenRAVE.

`<inertial>`
`  <pose>0 0 0.0714375 0 0 0</pose>`
`  <mass>5</mass>`
	`<mass type="custom">`
	`  <com>0 0 .0714375</com>`
	`  <total>5</total>`

La matriz de inercia en Gazebo se define dando los seis valores de sus elementos independientes (matriz simétrica). En OpenRAVE se indican los nueve elementos de la matriz, por filas.

`<inertia>`
`  <ixx>0.06908</ixx>`
`  <ixy>0.0000</ixy>`
`  <iyy>0.028705</iyy>`
`  <ixz>0.0000</ixz>`
`  <iyz>0.0000</iyz>`
`  <izz>0.073708</izz>`
`</inertia>`
`</inertial>`
	`<inertia>0.06908 0 0 0 0.028705 0 0 0 0.073708</inertia>`
	`</mass>`

Declaración de geometría con malla 3D como elemento visual y geometrías primitivas como colisión.

`<visual name="visual">`
`  <geometry>`
`  <mesh>`
`  <uri>model://cinder_block_2/meshes/cinder_block.dae</uri>`
`  </mesh>`
`  </geometry>`
`</visual>`
	`<Geom name="visual" type="sphere">`
	`  <radius>0</radius>`
	`  <render>meshes/cinder_block_2/cinder_block.iv</render>`
	`</Geom>`

El atributo **render** en OpenRAVE permite definir una geometría como no visible (sólo afecta a la colisión).

`<collision name="top">`
`  <pose>0 0 0.130175 0 0 0</pose>`
`  <geometry>`
`  <box>`
`  <size>0.19368008 0.396875 0.0254</size>`
`  </box>`
`  </geometry>`
`</collision>`
`</link>`
`</model>`
`</sdf>`
	`<Geom name="top" type="box" render="false">`
	`  <Translation>0 0 .130175</Translation>`
	`  <extents>.09684004 .1984375 .0127</extents>`
	`</Geom>`
	`</Body>`
	`</KinBody>`

- Tarea 5

  - Archivo de entorno

Fichero inicial en Gazebo

Nombre: drc_practice_task_5.world

```
<sdf version="1.4">
 <world name="trials_task_3">
  <!-- A ground plane -->
  <include>
   <uri>model://ground_plane</uri>
  </include>
  <!-- Ladder -->
  <include>
   <name>ladder</name>
   <uri>model://drc_practice_ladder</uri>
  </include>
 </world>
</sdf>
```

Fichero final en OpenRAVE

Nombre: drc_task_5.env.xml

```
<Environment>
  <!-- A ground plane -->

  <KinBody name="Floor"
file="models/ground_plane.kinbody.xml"/>

  <!-- Ladder -->

  <KinBody name="ladder"
file="models/task_5/drc_practice_ladder.ki
nbody.xml"/>

</Environment>
```

  - Archivo de modelo

Fichero inicial en Gazebo

Nombre modelo: drc_practice_ladder

Archivo: model.sdf

```
<sdf version="1.5">
 <model name="ladder">
  <static>true</static>
  <link name="link">
```

Fichero final en OpenRAVE

Nombre: drc_practice_ladder.kinbody.xml

```
<KinBody name="ladder">

 <Body name="link" type="static">
```

Definición de geometría cilíndrica.

```
   <collision
name="left_railing_landing_3">
```

```
  <Geom name="left_railing_landing_3"
type="cylinder" render="false">
```

El sistema de coordenadas local de un cilindro es distinto en Gazebo y en OpenRAVE. Para que la orientación del cilindro en el entorno sea la misma, es necesario aplicar una rotación extra de 90° alrededor del eje x al cilindro en OpenRAVE. Debido a la simetría del cilindro, el sentido del giro no es relevante.

Una rotación de 90° en el eje x en Gazebo equivale a un cilindro sin rotar en OpenRAVE.

```
    <pose>-0.433355 0.66933 3.81 1.5707 0
0</pose>
    <geometry>
     <cylinder>
      <radius>0.02413</radius>
      <length>0.73172</length>
     </cylinder>
    </geometry>
   </collision>
   <collision
name="left_railing_landing_upright_back">
```

```
   <Translation>-0.433355 0.66933
3.81</Translation>


   <radius>0.02413</radius>
   <height>0.73172</height>


  </Geom>
  <Geom
name="left_railing_landing_upright_back"
type="cylinder" render="false">
```

Un cilindro sin rotar en Gazebo ha de tener una rotación de 90° en el eje x en OpenRAVE.

```
    <pose>-0.433355 1.01088 3.27660 0 0
0</pose>

    <geometry>
     <cylinder>
      <radius>0.02413</radius>
```

```
   <Translation>-0.433355 1.01088
3.27660</Translation>
   <RotationAxis>1 0 0 90</RotationAxis>


   <radius>0.02413</radius>
```

```
      <length>1.0668</length>
     </cylinder>
    </geometry>
   </collision>
   <collision
name="left_railing_upright_4">
```

```
   <height>1.0668</height>


  </Geom>
  <Geom name="left_railing_upright_4"
type="cylinder" render="false">
```

Hay que sumar (o restar) al ángulo que gira el cilindro alrededor del eje x en Gazebo los 90° que corresponden a situar el cilindro en la posición equivalente en OpenRAVE. Por tanto, pasamos de una rotación de 59° en Gazebo a 59° - 90° = -31° en OpenRAVE.

```
    <pose>-0.433355 0.06990 2.80986
1.0341369 0 0</pose>

    <geometry>
     <cylinder>
      <radius>0.02413</radius>
      <length>0.57</length>
     </cylinder>
    </geometry>
   </collision>
```

```
   <Translation>-0.433355 0.06990
2.80986</Translation>
   <RotationAxis>1 0 0 -31</RotationAxis>


   <radius>0.02413</radius>
   <height>0.57</height>


  </Geom>
```

4.1.3 Conversión de mallas 3D

Como ya se ha comentado, para representar un elemento mediante una geometría compleja se recurre a ficheros externos que definen mallas tridimensionales. También se ha visto que los formatos que se utilizan en Gazebo son distintos de los admitidos en OpenRAVE. El resultado es que, para poder utilizar las mallas empleadas en Gazebo en los modelos en OpenRAVE, es necesario hacer una conversión de formato de los archivos.

La palabra *modelo* se usa en el campo de diseño en 3D con un matiz distinto al que se le da en Robótica. De aquí en adelante y hasta el final de este capítulo, se hablará de *modelos 3D* para indicar una geometría definida mediante mallas tridimensionales. Un *modelo 3D* (acepción en diseño 3D) sirve para la representación de un *modelo* (acepción en Robótica) del entorno de un robot.

Se parte de modelos 3D en formato COLLADA y se quieren exportar a alguno de los formatos admitidos por OpenRAVE: VRML u Open Inventor. Para realizar esta conversión, se recurre a la utilización de programas de modelado en 3D.

Los programas que se emplean para el cambio de formato son fundamentalmente dos: SketchUp y Blender. Estos programas permiten convertir ficheros de COLLADA a los formatos VRML y Open Inventor, respectivamente.

La razón principal para usar estos programas es, además de que proporcionan la funcionalidad deseada, que se trata de las herramientas que se han empleado en la creación de los ficheros de COLLADA, por lo que se puede esperar que la exportación a un nuevo formato sea más precisa y no se pierda ni deteriore información. Esto se puede ver en el texto de los archivos de COLLADA. En la descripción del archivo al inicio del código la etiqueta **authoring_tool** indica el software utilizado para la creación del modelo 3D definido. Se dan dos casos, que se reflejan en los fragmentos de código Fragmento de código 4.1 y Fragmento de código 4.2, y que muestran que los programas utilizados para el diseño de los modelos 3D son SketchUp y Blender.

```
<authoring_tool>SketchUp 8.0.15158</authoring_tool>
```

Fragmento de código 4.1. Herramienta de creación de modelo: SketchUp. Aparece en la descripción de los modelos de la Tarea 1, Tarea 5, y en el modelo wood_slats *de la Tarea 2.*

```
<authoring_tool>Blender 2.68.0 r58536</authoring_tool>
```

Fragmento de código 4.2. Herramienta de creación del modelo: Blender. Aparece en la descripción de los modelos de la Tarea 2 (excepto wood_slats*).*

Por las razones explicadas anteriormente, se va a utilizar para el cambio de formato en cada fichero la herramienta que se ha empleado para la creación del modelo 3D. Por las funcionalidades que ofrece cada programa, los modelos 3D exportados con SketchUp tendrán formato VRML, mientras que los generados con Blender estarán definidos por ficheros Open Inventor. A continuación se indican las principales consideraciones a tener en cuenta para la conversión mediante cada uno de estos programas.

- SketchUp

SketchUp es un producto de la empresa Trimble Navigation. Se trata de un programa de modelado en 3D orientado a las áreas de Arquitectura, Diseño, Construcción, Fabricación e Ingeniería [62].

SketchUp permite la importación de ficheros en formato COLLADA y su posterior exportación en formato VRML. Como ya se ha argumentado, se emplea SketchUp para la conversión de los modelos 3D generados con dicho programa.

Las indicaciones que se dan a continuación corresponden a la versión 14.0.4900, que es la que se ha empleado para la realización de este proyecto.

- Escalado en SketchUp

Los archivos de COLLADA creados con SketchUp están escalados según el sistema de medida británico, en pulgadas. La unidad empleada se indica en la descripción del modelo 3D, dentro de la etiqueta **unit**, como se ve en el Fragmento de código 4.3.

```
<unit meter="0.02539999969303608" name="inch" />
```

Fragmento de código 4.3. Unidades empleadas en los archivos de COLLADA. SketchUp

Los modelos 3D generados automáticamente a partir de éstos se definen también en pulgadas, por lo que requieren un escalado posterior en OpenRAVE para su definición en metros (el factor de conversión a aplicar es 0.0254). Ya se ha visto en el apartado correspondiente cómo se escalan modelos 3D, como recordatorio, se presenta el Fragmento de código 4.4.

```
<render>meshes/o_jersey_barrier.wrl 0.0254</render>
<collision>meshes/o_jersey_barrier.wrl 0.0254</collision>
```

Fragmento de código 4.4. Escalado en OpenRAVE: paso de pulgadas a metros

Otra posibilidad es aplicar el factor de escala directamente al modelo 3D generado, modificando manualmente el fichero VRML mediante un editor de texto e incluyendo el Fragmento de código 4.5 dentro de la etiqueta que da las transformaciones aplicadas al objeto.

```
scale 0.025400 0.025400 0.025400
```

Fragmento de código 4.5. Escalado en fichero VRML

- Origen en SketchUp

Al importar un modelo en SketchUp, éste no se sitúa automáticamente en la plantilla de modelado, sino que es necesario colocarlo de manera manual. La ubicación del modelo dentro de la plantilla determina el sistema de coordenadas local del mismo (y el punto respecto al cual se realizan las transformaciones), por lo que es necesario tener en cuenta una serie de

consideraciones al posicionarlo para que se mantenga el sistema de coordenadas del modelo inicial.

Para exportar los ficheros manteniendo el origen de coordenadas local se sigue el proceso descrito a continuación. Inicialmente, al cargar el modelo, el cursor del ratón está situado en el origen de coordenadas local del objeto (que no tiene que coincidir con el origen geométrico), como se aprecia en la Figura 4.1, de forma que éste se mueve solidariamente con el desplazamiento del cursor. Teniendo esto en cuenta, se puede desplazar el objeto hasta hacer coincidir el origen del modelo importado (cursor del ratón) con el origen de coordenadas definido en la plantilla de modelado. Para ello resulta de gran utilidad la herramienta de inferencia que ofrece SketchUp, que permite identificar el origen de coordenadas (punto *origin*) con exactitud como se muestra en la Figura 4.2.

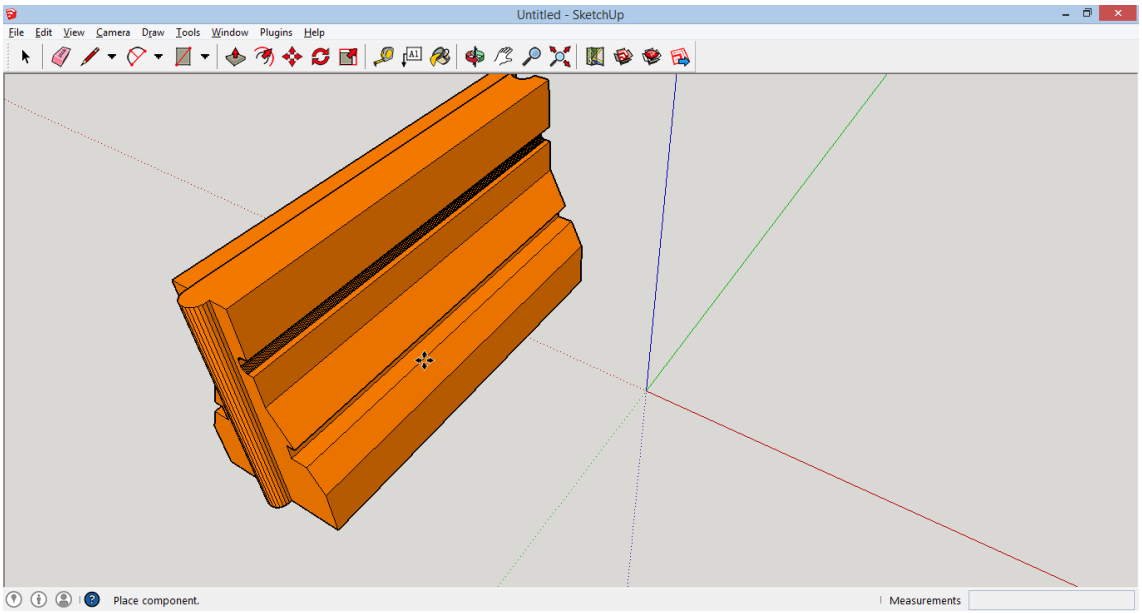

Figura 4.1. En SketchUp el cursor se sitúa inicialmente en origen local del modelo

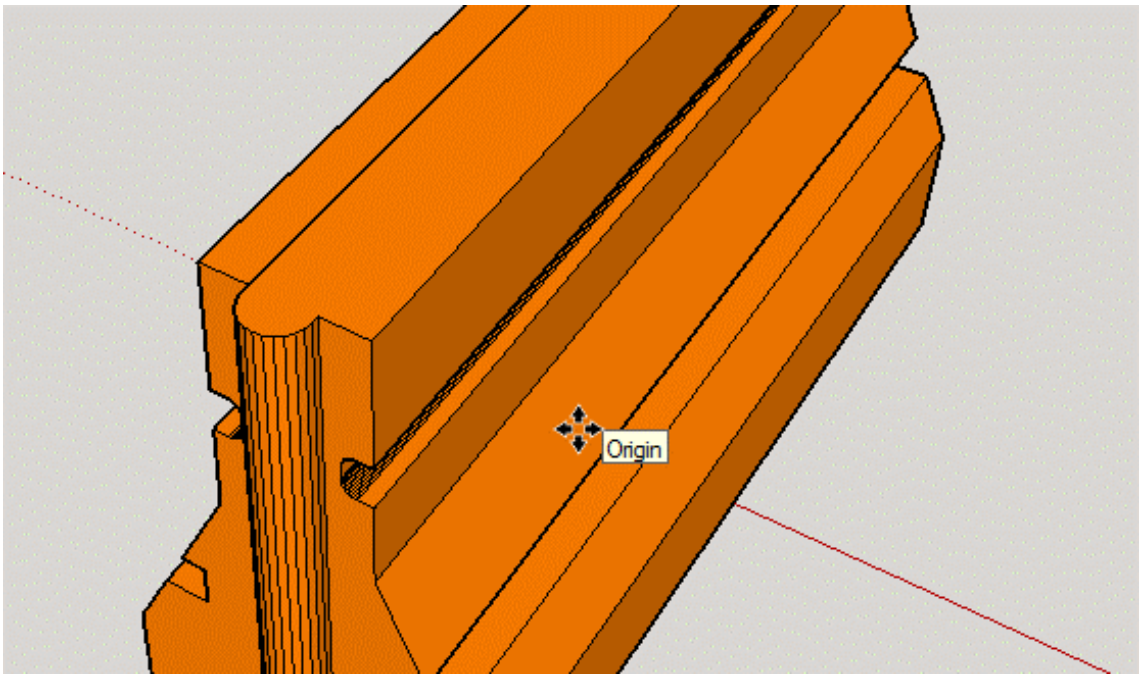

Figura 4.2. Inferencia de puntos en SketchUp

- Blender

Blender es un software libre de diseño en 3D. Fue creado por la Fundación Blender, una organización sin ánimo de lucro. Actualmente en su desarrollo colaboran voluntarios de todo el mundo [63].

Aunque en las versiones más recientes de Blender esta funcionalidad ha desaparecido, existen versiones antiguas que permiten la conversión de archivos COLLADA al formato Open Inventor admitido en OpenRAVE. En la documentación de Blender se indica que las versiones 2.42a, 2.45, 2.46 soportan la exportación a archivos Open Inventor [64], [65]. En la versión 2.46 no se ha conseguido encontrar esta funcionalidad, pero la versión 2.45 sí que la ofrece, y es la que se ha empleado en este proyecto para la conversión de los modelos correspondientes.

- Escalado en Blender

En los ficheros Open Inventor generados por Blender la escala de paso de pulgadas a metros ya está aplicada en el código del fichero, mediante una matriz de transformación como la reflejada en el Fragmento de código 4.6.

```
MatrixTransform
    {
      matrix
      0.025400 0.000000 0.000000 0.000000
      0.000000 0.025400 0.000000 0.000000
      0.000000 0.000000 0.025400 0.000000
      0.000000 0.000000 0.000000 1.000000
    }
```

Fragmento de código 4.6. Escalado con Blender

Por tanto, los ficheros exportados con Blender se pueden referenciar en el código de OpenRAVE sin necesidad de aplicar un factor de escala.

- Origen en Blender

Los modelos se colocan automáticamente en la plantilla al importarlos, por lo que no es necesario aplicar ninguna transformación manual para situar el sistema de coordenadas del objeto.

- Texturas en Blender

Al exportar un modelo mediante Blender, el fichero generado depende de un archivo **.png* que define la textura. Sin embargo, Blender no crea este archivo. Puesto que partimos de los ficheros de COLLADA para Gazebo, podemos copiar el archivo de texturas almacenado en la carpeta *textures* del modelo y renombrarlo adecuadamente para asociarlo al fichero Open Inventor creado por Blender.

4.2 VEHÍCULO

Se trata de un modelo que forma parte del entorno de la tarea 1, que por su complejidad merece ser estudiado en un capítulo aparte.

La estructura del modelo en Gazebo es la siguiente:

- *polaris_ranger_xp900*: carpeta donde se incluyen los ficheros que definen el modelo.
    - *model.config*: metadatos sobre el modelo.
    - *model.sdf*: descripción del modelo en formato SDF.
    - *meshes*: directorio con los ficheros de modelos 3D asociados.
        - *polaris.dae*: modelo 3D del vehículo en formato COLLADA.
    - *materials*: directorio con texturas y plugins asociados a los modelos 3D.
        - *scripts*: contiene el fichero *polaris.material*.
        - *textures*: contiene el fichero con la textura asociada al modelo.
            - *RangerXP900_Diffuse.png*

Para adaptar el modelo del vehículo a OpenRAVE es necesario:

- transformar el fichero *model.sdf* y generar un fichero **.kinbody.xml* con el formato propio de OpenRAVE,
- cambiar el formato del fichero *polaris.dae* a uno de los reconocidos por OpenRAVE.

En el caso de este elemento, por las características de los ficheros, se va a estudiar en primer lugar la conversión del fichero *polaris.dae* que define la malla 3D y a continuación la adaptación del formato XML del archivo *model.sdf*.

4.2.1 Malla 3D: *polaris.dae*

En el código del fichero *polaris.dae* que define el vehículo podemos ver que éste se compone de una serie de elementos individuales, etiquetados como **node** en el código, que dan lugar al vehículo completo. Es decir, se define el modelo completo como un conjunto de modelos más sencillos, que corresponden a los diferentes elementos que conforman el vehículo: ruedas, frenos, volante, chasis, etc. El formato COLLADA permite esta definición de un modelo global complejo como un conjunto de otros modelos de menor complejidad.

Para poder trabajar con el archivo en OpenRAVE es necesario descomponer el modelo del vehículo en sus elementos individuales. Es decir, a partir del fichero *polaris.dae* se va a generar un **.dae* independiente por cada elemento que compone el vehículo. Finalmente, estos ficheros de COLLADA individuales han de convertirse al formato VRML u Open Inventor para poder ser manipulados por OpenRAVE. Las razones por las que resulta necesario hacer esta descomposición son las siguientes:

1. Los elementos que componen el vehículo son accesibles de forma individual desde Gazebo, pero no desde OpenRAVE. OpenRAVE trata el modelo 3D completo definido en un mismo fichero como un sólido rígido. Para trabajar con los modelos por separado, es necesario definirlos mediante archivos individuales.
2. En Gazebo se aplican escalados no homogéneos a algunos de estos modelos 3D. En OpenRAVE sólo es posible aplicar escalados homogéneos, por lo que en estos casos será necesario escalar directamente los modelos 3D correspondientes modificando el fichero de COLLADA.
3. El vehículo definido en el archivo *polaris.dae* tiene la rueda delantera derecha mal colocada respecto a su posición relativa en el vehículo, y para recolocarla es necesario modificar la geometría asociada a este elemento.

- Creación de ficheros COLLADA individuales

Cada fichero **.dae* que define un modelo 3D individual se genera a partir del archivo *polaris.dae*. La estructura de etiquetas del archivo se ve en el Fragmento de código 4.7.

```
<COLLADA>
  <asset>
  <library_effects>
  <library materials>
  <library_geometries>
  <library_lights>
  <library_images>
  <library_visual_scenes>
  <scene>
```

Fragmento de código 4.7. Estructura de etiquetas de un archivo de COLLADA

La información específica relativa a la geometría y representación de cada elemento (**node**) se encuentra dentro de las etiquetas **library_geometries** y **library_visual_scenes**, respectivamente.

Para crear el archivo **.dae* correspondiente a un elemento se utiliza el mismo código del fichero *polaris.dae*, prescindiendo únicamente de los datos concretos asociados al resto de modelos 3D (en las etiquetas **library_geometries** y **library_visual_scenes**).

Para mantener cada elemento en la misma posición relativa que tiene en el modelo del vehículo completo es necesario conservar las transformaciones asociadas a dicho elemento así como las correspondientes a todos sus nodos padre (etiquetas con mayor jerarquía en el código). Se puede ver un ejemplo más adelante en el Fragmento de código 4.8.

Teniendo todo esto en cuenta se crean los ficheros de COLLADA correspondientes a los elementos individuales que componen el vehículo.

- Aplicación de factor de escala en ficheros COLLADA

Como ya se ha comentado, es necesario escalar aquellos modelos a los que se aplica un escalado heterogéneo en Gazebo. Esto se consigue incluyendo dentro de la etiqueta **library_visual_scenes**, en la descripción de la representación del elemento, la etiqueta **scale** con los valores de escalado correspondientes. En el Fragmento de código 4.8 se puede ver el escalado aplicado al elemento *Wheel_Front_Left*, que a su vez es hijo de los elementos *Body* y *Wheel_Front_Left*, que se conservan en el código porque las transformaciones de los elementos padre afectan al elemento hijo.

```
<library_visual_scenes>
  <visual_scene id="MaxScene">
    <node id="node-Body" name="Body">
      <node id="node-Brake_Front_Left" name="Brake_Front_Left">
        <translate>-52.75282 108.5968 35.51725</translate>
        <scale>0.08722734 0.08722734 0.08722734</scale>
        <node id="node-Wheel_Front_Left" name="Wheel_Front_Left">
          <scale>1.003700111 0.886200464 0.886200464</scale>
            ...
```

Fragmento de código 4.8. Escalado de un modelo en COLLADA

- Modificación posición rueda

Como se puede ver en la Figura 4.3, al cargar el fichero *polaris.dae* en SketchUp, la rueda delantera derecha del vehículo no está en la posición esperada[15].

[15] Esto es así en el fichero *polaris.dae* disponible con la versión de DRCSim descargada en marzo de 2014, y es posible que se haya solucionado en versiones posteriores.

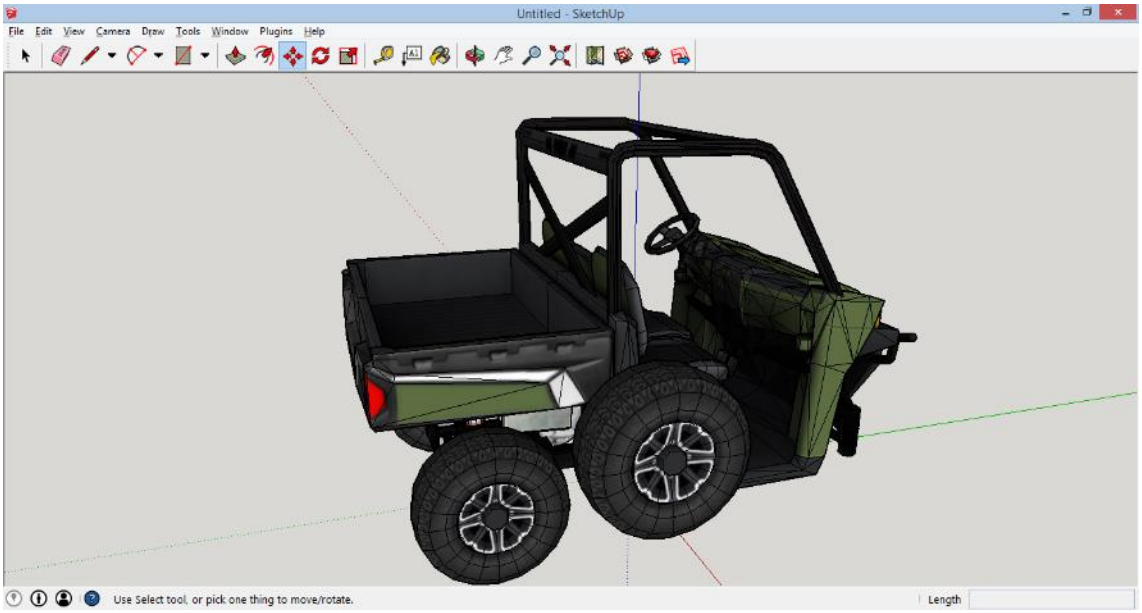

Figura 4.3. Representación del fichero inicial del vehículo en SketchUp

Es necesario modificar las coordenadas de los vértices que definen la geometría de dicha rueda en el fichero de COLLADA para obtener la posición adecuada. Para los cálculos se va emplear el programa Matlab.

Las coordenadas de un elemento (**node**) vienen definidas como ternas de valores (que representan puntos en el espacio) dentro de la etiqueta **library_geometries**. El modelo es el resultado de unir todos los puntos. Puesto que la forma geométrica de la rueda es correcta y lo único que es necesario modificar es la posición relativa en el modelo del vehículo, únicamente habrá que sumar o restar un mismo valor a todas las coordenadas de cada eje. Para calcular cuál es este valor se toma como referencia la rueda opuesta situada en el mismo eje (la cual se haya correctamente colocada).

En primer lugar, se calculan las coordenadas de la bounding box[16] de cada rueda (que vienen dadas por las coordenadas máxima y mínima en cada eje). En la Figura 4.4 se puede ver la bounding box correspondiente a una rueda. La diferencia entre las coordenadas de las bounding boxes será el valor que hay que sumar o restar a todos los puntos en cada eje. En los ejes y y z las coordenadas de la bounding box deben ser las mismas para ambas ruedas. En el eje x las ruedas deben estar separadas por un factor dado por la transformación aplicada.

[16] Bounding Box es lo que a veces se conoce como volumen delimitador o envolvente. Se trata de un ortoedro que envuelve tanto como sea posible a la selección.

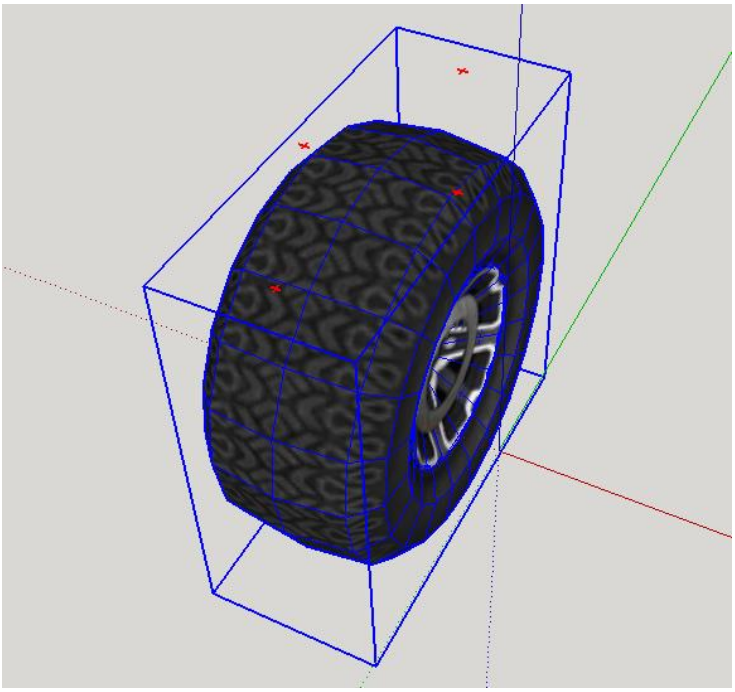

Figura 4.4. Bounding Box

En la Figura 4.5 se puede apreciar la posición relativa de las ruedas delanteras antes y después de modificar la geometría de la rueda derecha.

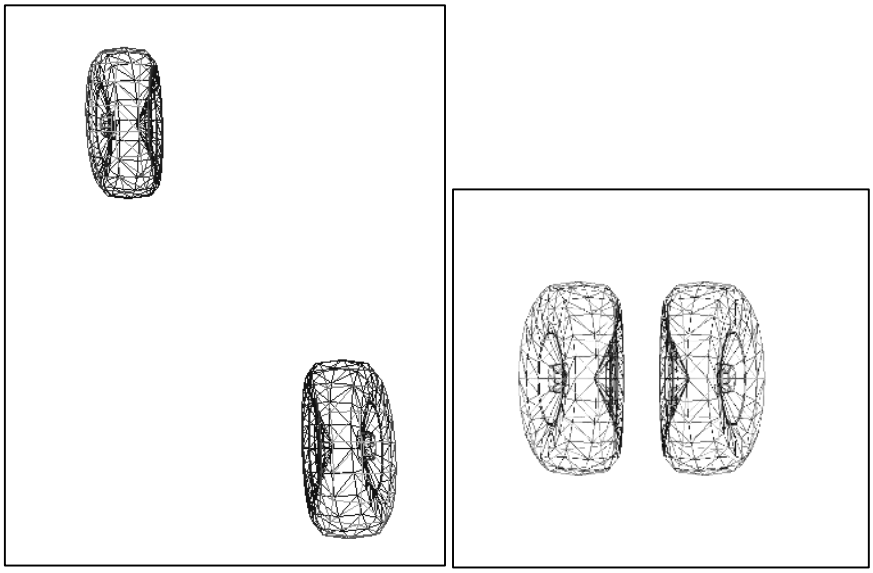

Figura 4.5. Representación en MeshLab en wireframe de la posición relativa de las ruedas antes y después de modificar las coordenadas de sus vértices

El resultado de aplicar esta modificación en los puntos que definen la geometría se ve en la Figura 4.6.

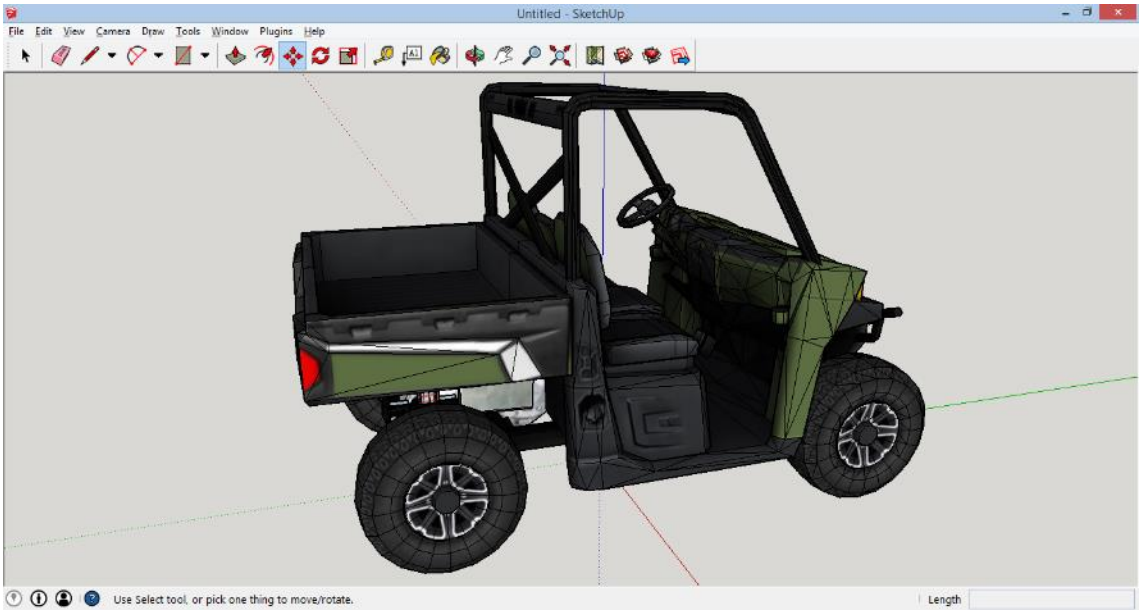

Figura 4.6. Representación del fichero del vehículo en SketchUp tras aplicar las modificaciones

- Conversión de ficheros COLLADA a formato para OpenRAVE

Es necesario convertir todos los ficheros generados en formato COLLADA al formato VRML u Open Inventor para permitir su interacción con OpenRAVE.

En este caso se realiza la conversión al formato Open Inventor con Blender, ya que el resultado es un vehículo con un mejor aspecto visual en el entorno.

4.2.2 Fichero XML: *model.sdf*

En este apartado se estudian las partes representativas del código y aquellas que representan alguna diferencia respecto a lo que se ha visto en el apartado anterior sobre la adaptación de código XML.

```
<sdf version="1.5">
 <model name="polaris_ranger_xp900">
```

```
<KinBody name="polaris_ranger_xp900">
```

Se trata de un modelo dinámico. En Gazebo se indica esta característica para el modelo completo, dando el valor **false** a la etiqueta **static**. En OpenRAVE hay que definir cada link como dinámico.

```
  <static>false</static>
  <link name="chassis">
   <inertial>
     ...
   </inertial>
   <visual name="visual">
    <pose>0 0 0 0 0 -1.570796</pose>
    <geometry>
```

```
<Body name="chassis" type="dynamic">
 <Mass type="custom">
    ...
 </Mass>
 <Geom name="visual" type="trimesh">
  <RotationAxis>0 0 1 -90</RotationAxis>
```

Para la representación del vehículo, se emplea el fichero *polaris.dae*.

```
     <mesh>
      <uri>model://polaris_ranger_xp900/
meshes/polaris.dae</uri>
```

La etiqueta **submesh** en Gazebo permite acceder, dentro del fichero *polaris.dae*, a los elementos individuales que conforman el vehículo completo, indicados en el fichero de COLLADA mediante la etiqueta **node**. En OpenRAVE no se dispone de ningún elemento con esta

función, como alternativa, se utilizan archivos independientes para definir los distintos elementos del coche.

```
      <submesh>
```

Dentro de **submesh**, se definen las etiquetas **name** y **center**. Mediante la etiqueta **name** se indica el elemento a incluir. Los nodos tienen un nombre asignado dentro del fichero de COLLADA, y en este campo en Gazebo se hace referencia a dicho nombre.

```
       <name>Body</name>
```

```
   <render>meshes/polaris/Body.iv
0.01</render>
```

La etiqueta **center** define dónde se sitúa el origen local del elemento (que es lo que se tiene en cuenta a la hora de aplicar las transformaciones al modelo). El valor **false** indica que el elemento individual (submesh) se coloca referido al origen global del modelo del que forma parte. Si se da a **center** el valor **true**, se indica que el origen local del elemento se va a situar en su centro geométrico, entendido como el centro geométrico de su "bounding box" [66].

En OpenRAVE no existe ninguna etiqueta con la misma función que **center**. Los ficheros de COLLADA se generan manteniendo la transformación asociada a cada elemento, por lo que ya están en la posición adecuada, no es necesario aplicar transformaciones. El modelo completo está girado 90° respecto a la posición que debe ocupar en el entorno, y ésta es la única transformación que hay que aplicar a los modelos 3D referenciados en OpenRAVE.

```
       <center>false</center>
      </submesh>
     </mesh>
    </geometry>
```

En Gazebo se definen en el fichero SDF las texturas asociadas al modelo. En OpenRAVE, el modelo 3D lleva ya las texturas asociadas.

```
    <material>
     <script>
      <uri>model://polaris_ranger_xp900/
materials/scripts</uri>
      <uri>model://polaris_ranger_xp900/
materials/textures</uri>
      <name>PolarisXP900/Diffuse</name>
     </script>
    </material>
   </visual>
```

```
  </Geom>
```

Se incluye un sensor en el vehículo que permite detectar cuándo se sube el robot. Como ya se ha visto en el Capítulo 3, en el apartado de Sensores, no es posible portar un sensor de tipo contacto a OpenRAVE, ya que no se dispone de sensores funcionales de este tipo. Para poder realizar la adaptación sería necesario en primer lugar definir el sensor adecuado utilizando los plugins disponibles en OpenGRASP.

```
   <sensor name="seat_contact"
type="contact">
    <contact>
     <collision>mud_seat</collision>
    </contact>
   </sensor>
   <visual name="cargo_visual">
```

```
  <Geom name="cargo_visual" type="trimesh">
```

Las transformaciones que se hacen para los modelos 3D que componen el coche en Gazebo no son necesarias para los modelos 3D creados para OpenRAVE. Por la forma en que se han generado los mismos, explicada anteriomente, sólo hay que aplicar una rotación de 90° alrededor del eje x.

```
    <pose>-1.0 0 1.03230 0 0 -1.5707</pose>
```

```
   <RotationAxis>0 0 1 -90</RotationAxis>
```

```
    <geometry>
     <mesh>
      <uri>model://polaris_ranger_xp900/
meshes/polaris.dae</uri>
      <submesh>
       <name>Bed</name>
```

```
   <render>meshes/polaris/Bed.iv
0.01</render>
```

La etiqueta **center** tiene en este caso el valor **true**, por tanto el modelo 3D en Gazebo tiene su origen local en su centro geométrico.

```
       <center>true</center>
      </submesh>
     </mesh>
    </geometry>
   </visual>
  </link>
  <link name="front_left_wheel">
```

```
  </Geom>
 </Body>
 <Body name="front_left_wheel"
type="dynamic">
```

En Gazebo se aplica una transformación al link completo. En OpenRAVE la transformación no debe afectar a las geometrías definidas mediante los modelos 3D. Por tanto, se aplica individualmente a las geometrías que corresponda.

```
   <pose>1.20223 0.71562 0.34697 -1.52 0.0
0.0</pose>
   <collision name="collision">
```

```
  <Geom type="cylinder" render="false">
```

Se aplican la traslación y rotación al elemento de colisión, ya que está representado mediante una geometría propia de OpenRAVE.

```
                                    <Translation>1.20223 0.71562
                                    0.34697</Translation>
                                    <RotationAxis>1 0 0 3</RotationAxis>
    <geometry>
     <cylinder>
      <radius>0.3175</radius>       <radius>0.3175</radius>
      <length>0.2794</length>       <height>0.2794</height>
     </cylinder>
    </geometry>
   </collision>                     </Geom>
   <visual name="tire_visual">      <Geom name="tire_visual" type="trimesh">
```

El elemento visual está definido mediante un modelo 3D, por lo que no hay que aplicar las transformaciones. Únicamente se aplica la rotación de 90° que corresponde a todos los modelos.

```
    <pose>0 0 0 3.14159 1.570796 0</pose>
    <geometry>
     <mesh>
      <uri>model://polaris_ranger_xp900/
meshes/polaris.dae</uri>
```

```
   <RotationAxis>0 0 1 -90</RotationAxis>
```

En Gazebo se puede definir un factor de escala distinto para cada eje. En OpenRAVE esto no es posible, lo que se hace es aplicar el escalado directamente al modelo 3D, como se ha visto en el apartado 4.2.1.

```
       <scale>1.003700111 0.886200464
0.886200464</scale>
       <submesh>
        <name>Wheel_Front_Left</name>
        <center>true</center>
       </submesh>
      </mesh>
     </geometry>
    </visual>
   </link>
   <link name="gas_pedal">
    <pose>0.63 0.10 0.58 0 0 0</pose>
```

```
   <render>meshes/polaris/Wheel_Front_Left
_scaled.iv 0.01</render>
```

```
  </Geom>
 </Body>
 <Body name="gas_pedal" type="dynamic">
```

Mediante la etiqueta **gravity** se indica en Gazebo que el link *gas_pedal* no va a estar afectado por la gravedad. No hay equivalente en OpenRAVE, el pedal se sostiene por su enlace con el resto del vehículo.

```
    <gravity>false</gravity>
    <inertial>
      ...
    </inertial>
    <collision name="gas_pedal_collision">
      ...
    </collision>
    <visual name="visual">
      ...
    </visual>
  </link>
  <link name="FNR_switch">
```

```
  <Mass type="custom">
     ...
  </Mass>
  <Geom name="gas_pedal_collision"
type="box" render="false">
     ...
  </Geom>
  <Geom type="trimesh">
     ...
  </Geom>
 </Body>
 <Body name="FNR_switch" type="dynamic">
```

El cuerpo no depende de mallas 3D, está definido únicamente con geometrías primitivas, por lo que se aplica la transformación en OpenRAVE al igual que en Gazebo.

```
    <pose>0.54 0.1 1.15 0 0.25 0</pose>
    <collision name="FNR_switch">
     <geometry>
      <box>
       <size>0.02 0.04 0.08</size>
      </box>
     </geometry>
    </collision>
    <visual name="FNR_switch_F">
```

```
  <Translation>0.54 0.1 1.15</Translation>
  <RotationAxis>0 1 0 14</RotationAxis>
  <Geom name="FNR_switch_F" type="box">

   <extents>0.01 0.02 0.04</extents>
```

Se puede definir la transparencia u opacidad de un elemento.

```
    <transparency>0.0</transparency>
    <geometry>
     <box>
      <size>0.02 0.04 0.08</size>
     </box>
    </geometry>
    <material>
     <script>
      <uri>model://polaris ranger ev/
materials/scripts</uri>
      <uri>model://polaris_ranger_ev/
materials/textures</uri>
      <name>FNR switch F</name>
     </script>
    </material>
   </visual>
  </link>
  <!-- Joints -->
```

```
   <transparency>0.0</transparency>

   <diffuseColor>0.443 0.718
0.38</diffuseColor> <!-- Green -->

  </Geom>
 </Body>
 <!-- Joints -->
```

Declaración de un enlace tipo bisagra.

```
  <joint type="revolute"
name="front_left_steering_joint">
   <child>front_left_wheel_steering_block
</child>
   <parent>chassis</parent>
   <axis>
    <xyz>0 0 1</xyz>
    <limit>
     <lower>-0.7727</lower>
     <upper>0.7727</upper>
    </limit>
```

```
 <Joint type="hinge"
name="front_left_steering_Joint">
  <Body>front_left_wheel_steering_block
</Body>
  <Body>chassis</Body>

  <axis>0 0 1</axis>

  <limitsrad>-0.7727 0.7727</limitsrad>
```

La etiqueta **use_parent_model_frame** en Gazebo indica que el giro o desplazamiento se da respecto al link definido como padre. Equivale a dar a **offsetfrom** en OpenRAVE el valor del que es el link padre en Gazebo.

```
    <use_parent_model_frame>true</use_
parent_model_frame>
   </axis>
  </joint>
```

```
  <offsetfrom>chassis</offsetfrom>

 </Joint>
```

```
  <joint type="revolute"
name="rear_right_wheel_joint">
```

```
 <Joint type="hinge"
name="rear_right_wheel_Joint">
```

Mediante la etiqueta **pose** o **anchor** se define la posición del punto de giro. La traslación se hace respecto al link hijo en Gazebo y respecto al cuerpo 0 en OpenRAVE (el que primero se indica mediante las etiquetas **Body**). La rotación no se tiene en cuenta.

```
   <pose>0.0 0.0 -0.1 0 0 0</pose>
   <child>rear_right_wheel</child>
   <parent>chassis</parent>
   <axis>
    <xyz>0 1 -0.05</xyz>
    <use_parent_model_frame>true</use_
parent_model_frame>
   </axis>
  </joint>
```

```
  <anchor>0.0 0.0 -0.1</anchor>
  <Body>rear_right_wheel</Body>
  <Body>chassis</Body>

  <axis>0 1 -0.05</axis>
  <offsetfrom>chassis</offsetfrom>

 </Joint>
```

Definición de joint tipo prismático.

```
  <joint name="gas_joint" type="prismatic">
   <child>gas_pedal</child>
   <parent>chassis</parent>
   <axis>
    <xyz>1.000000 0.000000 -1.000000</xyz>
    <limit>
     <lower>0.00</lower>
     <upper>0.08</upper>
    </limit>
    <use_parent_model_frame>true</use_
parent_model_frame>
   </axis>
  </joint>
 </model>
</sdf>
```

```
 <Joint name="gas_joint" type="slider">
  <Body>gas_pedal</Body>
  <Body>chassis</Body>

  <axis>1.000000 0.000000 -1.000000</axis>

  <limits>0.00 0.08</limits>

  <offsetfrom>chassis</offsetfrom>

 </Joint>
</KinBody>
```

4.3 ROBOT ATLAS

La conversión del robot tiene un interés especial puesto que la finalidad del proyecto es proporcionar un entorno en el que poder simular el comportamiento del robot y desarrollar algoritmos de locomoción y estabilidad. A partir del robot Atlas, que está orientado a este tipo de tareas, entre otras, es posible desarrollar este tipo de algoritmos. Por esto es importante adaptar el robot correctamente y conseguir un modelo funcional.

El robot Atlas está definido en Gazebo en formato URDF mediante ficheros **.xacro*. Ya se ha visto en el Capítulo 3 que la utilización de ficheros **.xacro* permite describir robots complejos, enlazando unos ficheros con otros mediante el uso de macros. La estructura y relación (simplificada) entre los ficheros que definen el robot se puede ver en el siguiente esquema:

- *atlas_v3_sandia_hands.urdf.xacro*: fichero principal que referencia y relaciona el resto de archivos dependientes que describen el robot.
    - *atlas_v3.urdf*: descripción del robot en formato URDF, exceptuando las manos y la cabeza.
    - *sandia_hand_simple_shapes.urdf.xacro*: fichero principal de descripción de las manos del robot, contiene las macros para descripción de los dedos y las palmas. Dentro de este archivo encontramos las siguientes macros:
        - *finger*: plantilla para definir un dedo, dando valores a los parámetros permite definir los distintos dedos de la mano del robot.
        - *sandia_hand*: plantilla para la descripción de una mano, incluye las macros de cada dedo, los valores que se da a los parámetros determinan si se trata de la mano derecha o izquierda.
    - *multisense_sl_v3.urdf*: descripción de la cabeza y sensores asociados a ésta en formato URDF.

4.3.1 Conversión automática

Para adaptar el robot a OpenRAVE, se puede recurrir a las herramientas de las que se ha hablado en el Capítulo 3, que permiten la exportación automática de robots en formato URDF al formato COLLADA válido en OpenRAVE.

En primer lugar, como en la definición de Atlas se utilizan ficheros **.xacro*, hay que generar a partir de ellos el fichero URDF correspondiente (ya que éste es el formato que permite la conversión a COLLADA). Como ya se ha comentado, el programa xacro disponible en ROS realiza automáticamente esta conversión, mediante la instrucción:

```
rosrun xacro xacro.py model.xacro > model.urdf
```

Donde *model.xacro* representa el fichero raíz que carga el robot. En este caso el fichero a convertir es: *atlas_v3_sandia_hands.urdf.xacro*, y está ubicado en la carpeta: */opt/ros/hydro/share/Atlas_description/robots*. En *model.urdf* se indica el directorio y el nombre del nuevo fichero URDF que contendrá la descripción completa del robot. Por ejemplo, para crear el robot *atlas.urdf* en la carpeta *Documents* la instrucción que habría que emplear sería:

```
rosrun xacro xacro.py /opt/ros/hydro/share/Atlas_description/robots/at
las_v3_sandia_hands.urdf.xacro > ~/Documents/atlas.urdf
```

A partir del fichero *atlas.urdf* generado, se puede crear un archivo en formato COLLADA utilizando la herramienta `collada_urdf` disponible en ROS. Para realizar esta conversión se emplea la siguiente instrucción:

```
rosrun collada_urdf urdf_to_collada model.urdf model.dae
```

En este caso, *model.urdf* representa el archivo que acabamos de generar conteniendo la descripción completa del robot (generado a partir de xacros), y *model.dae* es el archivo de destino. Siguiendo el ejemplo anterior, la instrucción a emplear para crear el archivo *atlas.dae* sería, una vez dentro de la carpeta *Documents*:

```
rosrun collada_urdf urdf_to_collada atlas.urdf atlas.dae
```

Es importante tener en cuenta algunas limitaciones en el funcionamiento del paquete `collada_urdf`:

- Esta herramienta no soporta texturas, por lo que se pierden en la conversión.
- Sólo permite exportar geometrías de tipo malla, por lo que, si existen geometrías primitivas en la descripción del robot, desaparecen tras la conversión.

En este caso, como resultado de la conversión, se obtiene un robot totalmente funcional y que mantiene las características que interesan relacionadas con las habilidades de locomoción. Las geometrías de los dedos no se exportan, ya que están representados mediante geometrías cilíndricas. Sin embargo, de momento no hay interés en trabajar con habilidades de manipulación, por lo que esto no afecta a la funcionalidad del robot.

Como se trata de un fichero en formato COLLADA, los programas de modelado en 3D pueden reconocerlo y cargarlo. En la Figura 4.7 se puede ver la representación del modelo en Blender. Se puede apreciar la posición de las articulaciones y los sensores visuales situados en la cabeza del robot.

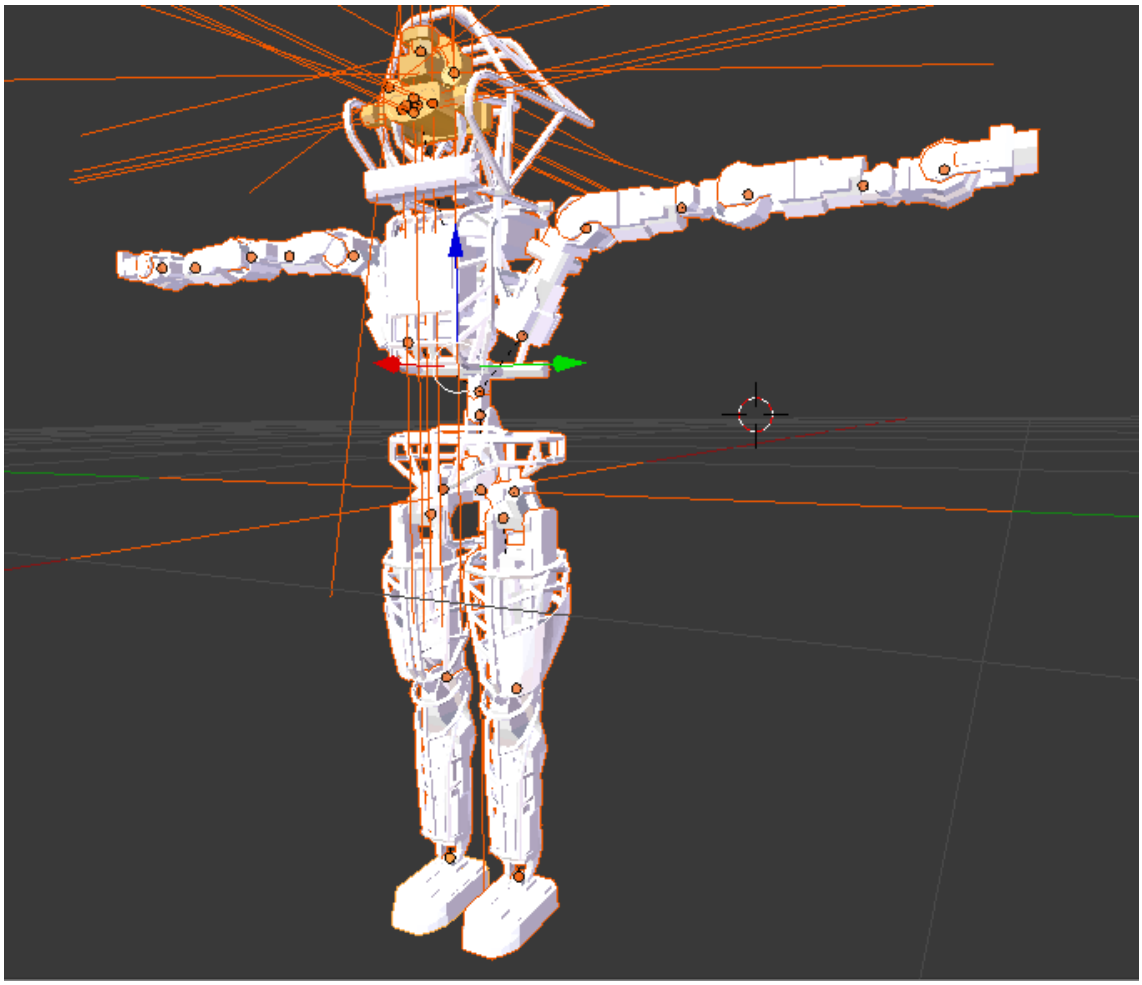

Figura 4.7. Representación del robot Atlas en Blender

Puesto que cumple con los objetivos, para la realización de este proyecto se ha optado por la conversión automática. Sin embargo, también es posible realizar la adaptación del robot de manera manual, que puede tener interés para proyectos futuros en que se trabaje con habilidades de manipulación.

4.3.2 Conversión manual

Conociendo la estructura de archivos que definen el robot y la conversión entre el formato URDF (muy similar al formato SDF empleado en Gazebo) y el XML propio de OpenRAVE es posible crear manualmente un archivo **.robot.xml* que describa el robot en OpenRAVE.

La dependencia de archivos del robot en OpenRAVE podría ser la siguiente:

- *atlas_v3_sandia_hands.robot.xml*: fichero principal que referencia y relaciona el resto de archivos dependientes que describen el robot.
    - *atlas_v3.robot.xml*: descripción del robot en formato XML, exceptuando las manos y la cabeza.
    - *sandia_hand_simple_shapes.robot.xml*: descripción de las manos del robot, a partir de los ficheros que describen las palmas y los dedos, es necesario convertir las macros a formato XML simple.
        - *right_sandia_hand.robot.xml*: descripción de la mano derecha, adaptando la macro *sandia_hand* sustituyendo los parámetros correspondientes al lado derecho e incluyendo la descripción de los dedos correspondientes a la mano derecha según la macro *finger*.
        - *left_sandia_hand.robot.xml*: descripción de la mano izquierda, adaptando la macro *sandia_hand* sustituyendo los parámetros correspondientes al lado izquierdo e incluyendo la descripción de los dedos correspondientes a la mano derecha según la macro *finger*.
    - *multisense_sl_v3.robot.xml*: descripción de la cabeza y sensores asociados a ésta en formato XML.

También es necesario convertir los ficheros de COLLADA que representan las geometrías del robot a un formato adecuado para OpenRAVE.

Este trabajo resulta muy costoso pero es posible, y tiene interés si se quiere obtener una conversión precisa de todos los elementos del robot, ya que en la conversión automática se pierden algunas características.

5 RESULTADOS. ENTORNOS EN OPENRAVE

Como resultado del proceso de adaptación de formatos quedan definidos en OpenRAVE los entornos y el robot de las pruebas de DRC, diseñados inicialmente para Gazebo. Una vez que se han definido los entornos para OpenRAVE, ha de comprobarse que el resultado es el esperado, es decir, que dichos entornos tienen características similares a los entornos en Gazebo y que el robot es funcional.

En este apartado se va a realizar la comparación gráfica de los entornos adaptados a OpenRAVE con los entornos iniciales creados para Gazebo. Para poder comparar es necesario cargar los entornos en cada simulador. Se habla primero brevemente sobre el proceso para realizar las simulaciones y a continuación se presentan imágenes del resultado de las mismas.

5.1 SIMULACIÓN DE LOS ENTORNOS

Para realizar la comparación entre los entornos se van a cargar los mismos mediante el simulador correspondiente, tanto en Gazebo como en OpenRAVE. En Gazebo se utiliza el simulador DRCSim; en OpenRAVE, Python.

5.1.1 Gazebo. Simulador DRCSim

Para lanzar una simulación en Gazebo se utiliza el comando `roslaunch`. Los entornos de las pruebas DRC se encuentran asociados al paquete `drcsim_gazebo` [67]. Por ejemplo, para cargar el entorno de la tarea 1 de DRC, se emplea la instrucción:

```
roslaunch drcsim_gazebo drc_practice_task_1.launch
```

El fichero *drc_practice_task_1.launch* contiene toda la información necesaria para cargar el entorno y lanzar la simulación.

De esta misma forma se cargan los entornos asociados a las pruebas 2 y 5. También es posible lanzar una simulación en la que se representa el robot Atlas en un entorno vacío, que correspondería a cargar el fichero *atlas.launch* [67].

5.1.2 OpenRAVE. Lenguaje Python

Para cargar los entornos en OpenRAVE se crean ficheros **.py*, que contienen la información necesaria sobre el entorno y modelos que se incluyen en la simulación, así como parámetros de la simulación. Estos ficheros se pueden ejecutar directamente mediante Python.

Python es un lenguaje de programación de código abierto desarrollado por la Python Software Foundation [68]. En este proyecto Python es el lenguaje que se emplea para interactuar con OpenRAVE y simular los entornos. A continuación se van a resumir algunas funciones de Python que se utilizan para simular entornos en OpenRAVE: cargar un entorno, incluir un robot o un objeto en el entorno y lanzar o detener la simulación.

- Cargar un entorno

Para cargar un entorno en OpenRAVE se emplea el Fragmento de código 5.1 [69]. Donde *worlds/drc_task_1.env.xml* se sustituye por el directorio y nombre del fichero que correspondan el entorno de interés.

```
env = Environment() # create openrave environment
env.SetViewer('qtcoin') # attach viewer (optional)
env.Load('worlds/drc_task_1.env.xml') # load a simple scene
```

Fragmento de código 5.1. Carga de un entorno en OpenRAVE mediante Python

- Incluir un robot en el entorno

Se puede incluir un robot en el entorno utilizando funciones de Python. Existen dos funciones que cumplen este objetivo: *env.Load* y *env.ReadRobotXMLFile*. Las instrucciones correspondientes en Python se muestran, respectivamente, en los fragmentos de código Fragmento de código 5.2 y Fragmento de código 5.3 [70].

```
env.Load('worlds/robots/atlas_task_1.robot.xml')
```

Fragmento de código 5.2. Carga de un robot en el entorno en OpenRAVE mediante la función env.Load de Python

```
robot = env.ReadRobotXMLFile('worlds/robots/atlas_task_1.robot.xml')
env.Add(robot)
```

Fragmento de código 5.3. Carga de un robot en el entorno en OpenRAVE mediante la función env.ReadRobotXMLFile de Python

- Incluir un objeto en el entorno

Para cargar a través de Python un objeto en el entorno se emplea la función *env.ReadKinBodyXMLFile*, como se ve en el Fragmento de código 5.4 [70].

```
target = env.ReadKinBodyXMLFile('worlds/models/polaris.kinbody.xml')
env.Add(target)
```

Fragmento de código 5.4. Carga de un objeto en el entorno en OpenRAVE mediante Python

- Lanzar y detener la simulación

Para iniciar la simulación con un paso de 0.001 se utiliza la instrucción [69]:

```
env.StartSimulation(0.001)
```

Fragmento de código 5.5. Iniciar la simulación en OpenRAVE mediante Python

Para detenerla:

```
env.StopSimulation
```

Fragmento de código 5.6. Detener la simulación en OpenRAVE mediante Python

Si al lanzar la simulación no se indica el paso de tiempo, se toma el valor por defecto (0.01).

Recapitulando con un ejemplo, para cargar el entorno de la tarea 1, incluir el robot Atlas y el vehículo y lanzar la simulación con un tiempo de paso de 0.001 mediante el fichero de Python, se emplearía el código reflejado en el Fragmento de código 5.7.

```
from openravepy import *
import time

env = Environment() # create openrave environment
env.SetViewer('qtcoin') # attach viewer (optional)
env.Load('worlds/drc_task_1.env.xml') # load a simple scene
# Add robot Atlas to the environment in the position for task_1
robot = env.ReadRobotXMLFile('worlds/robots/atlas_task_1.robot.xml')
env.Add(robot)
# Add polaris
target = env.ReadKinBodyXMLFile('worlds/models/polaris.kinbody.xml')
env.Add(target)
# Start simulation with time step=0.001
env.StartSimulation(0.001)

raw_input()
while True:
  time.sleep(5)
```

Fragmento de código 5.7. Código de Python para lanzar la simulación del entorno de la tarea 1 en OpenRAVE

La función *time.sleep* se emplea para evitar que se cierre el visualizador.

Utilizando los códigos necesarios para simular cada entorno en OpenRAVE y los ficheros correspondientes en Gazebo es posible comparar los resultados obtenidos en ambos casos.

5.2 COMPARACIÓN GRÁFICA DE LOS ENTORNOS

A continuación se muestra el resultado de lanzar las simulaciones para los entornos de las tareas 1, 2 y 5 con las que se ha trabajo.

5.2.1 Tarea 1

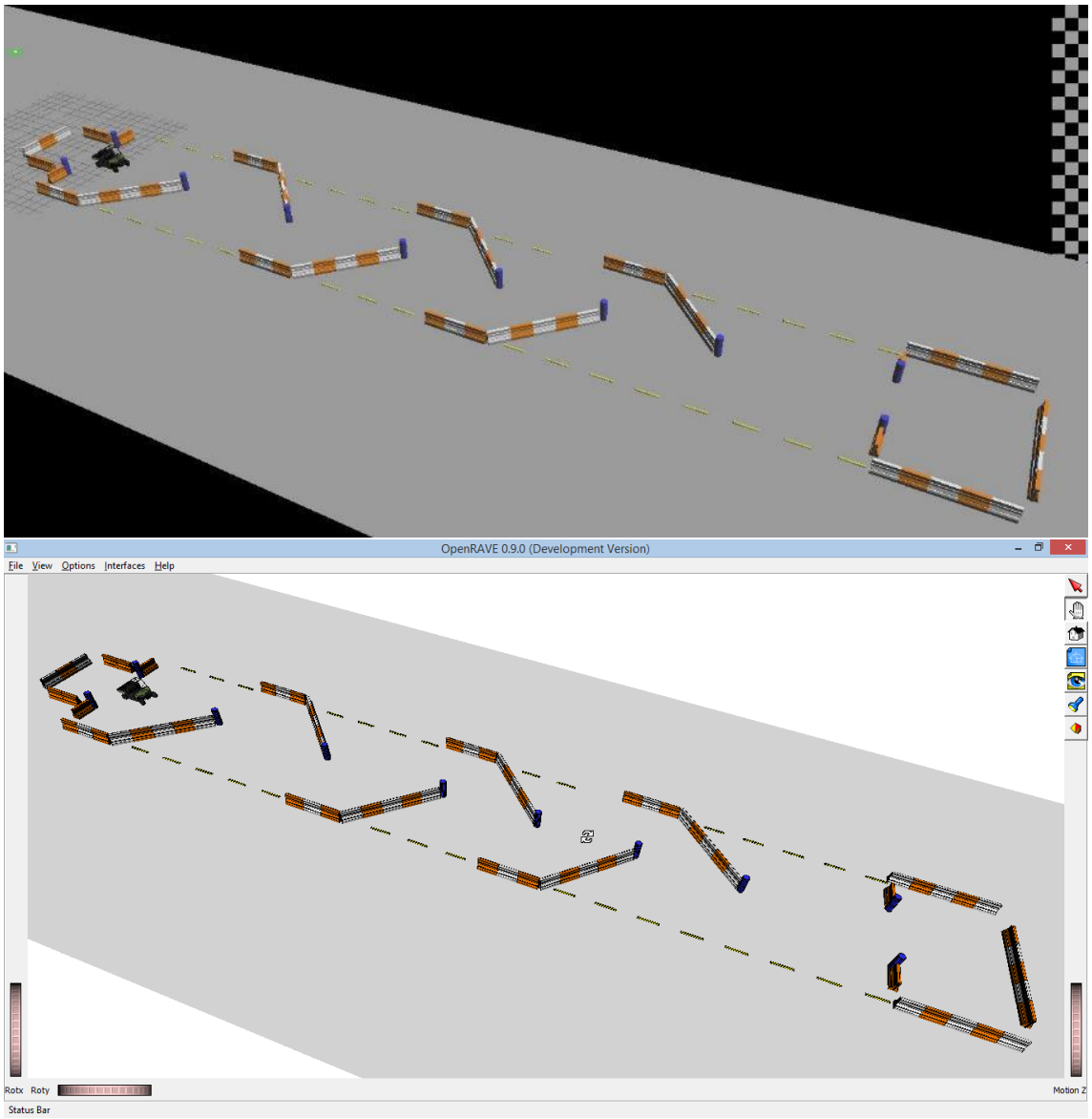

Figura 5.1. Entorno de la tarea 1 en Gazebo (parte superior) y en OpenRAVE (parte inferior) sin el robot Atlas

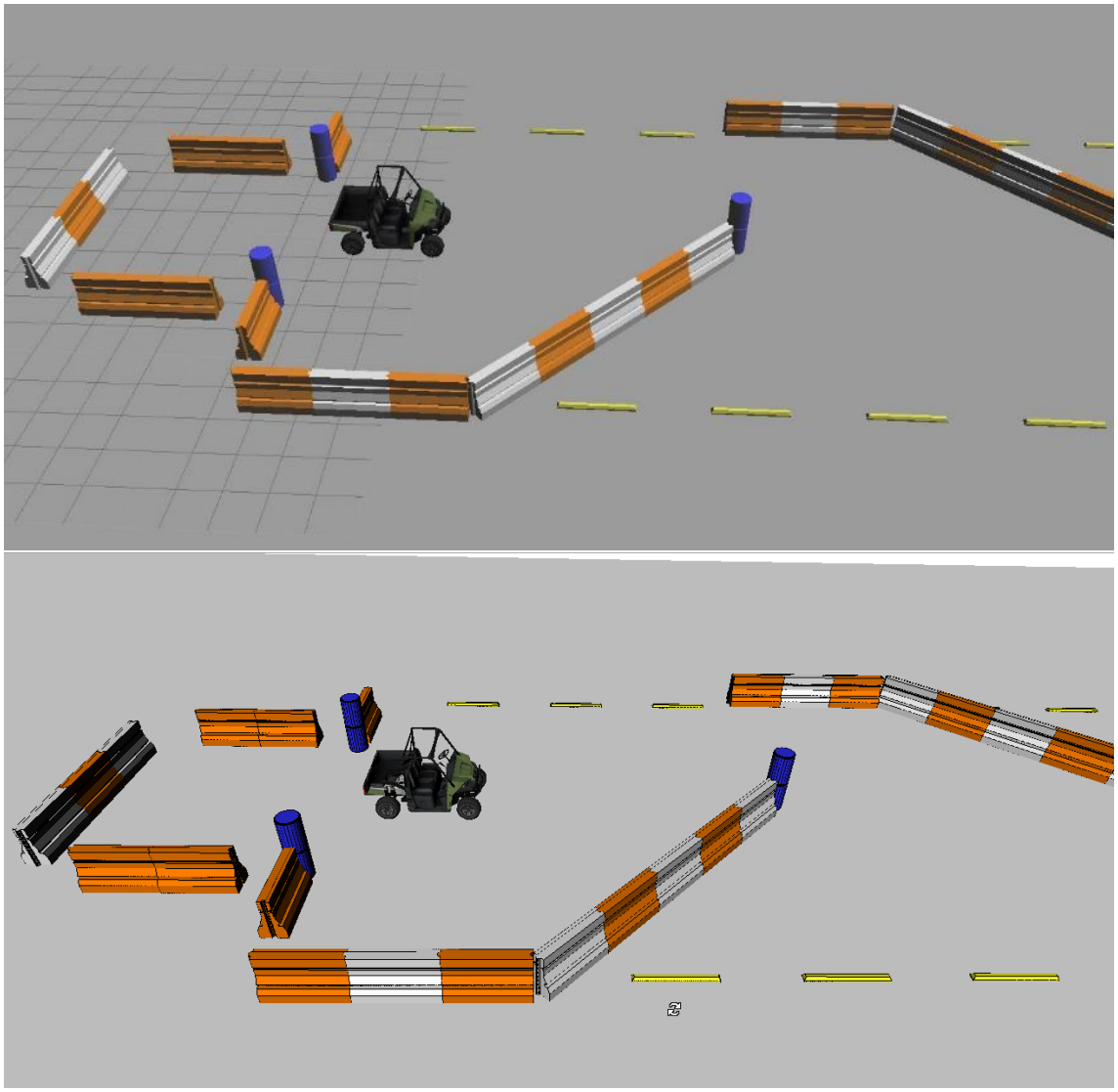

Figura 5.2. Zoom en el entorno de la tarea 1 en Gazebo (parte superior) y en OpenRAVE (parte inferior)

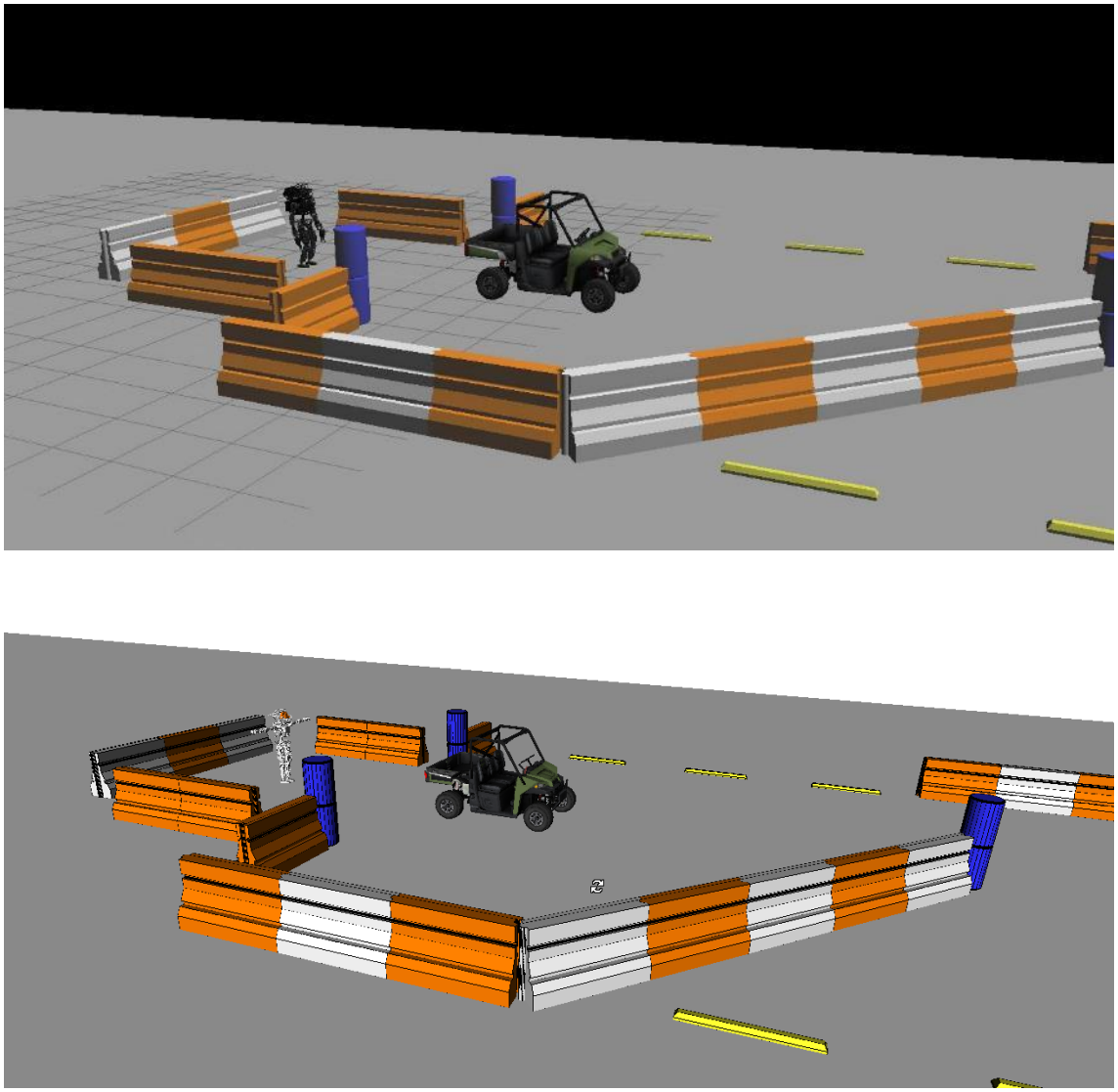

Figura 5.3. Entorno de la tarea 1 en Gazebo (parte superior) y en OpenRAVE (parte inferior) con el robot Atlas

5.2.2 Tarea 2

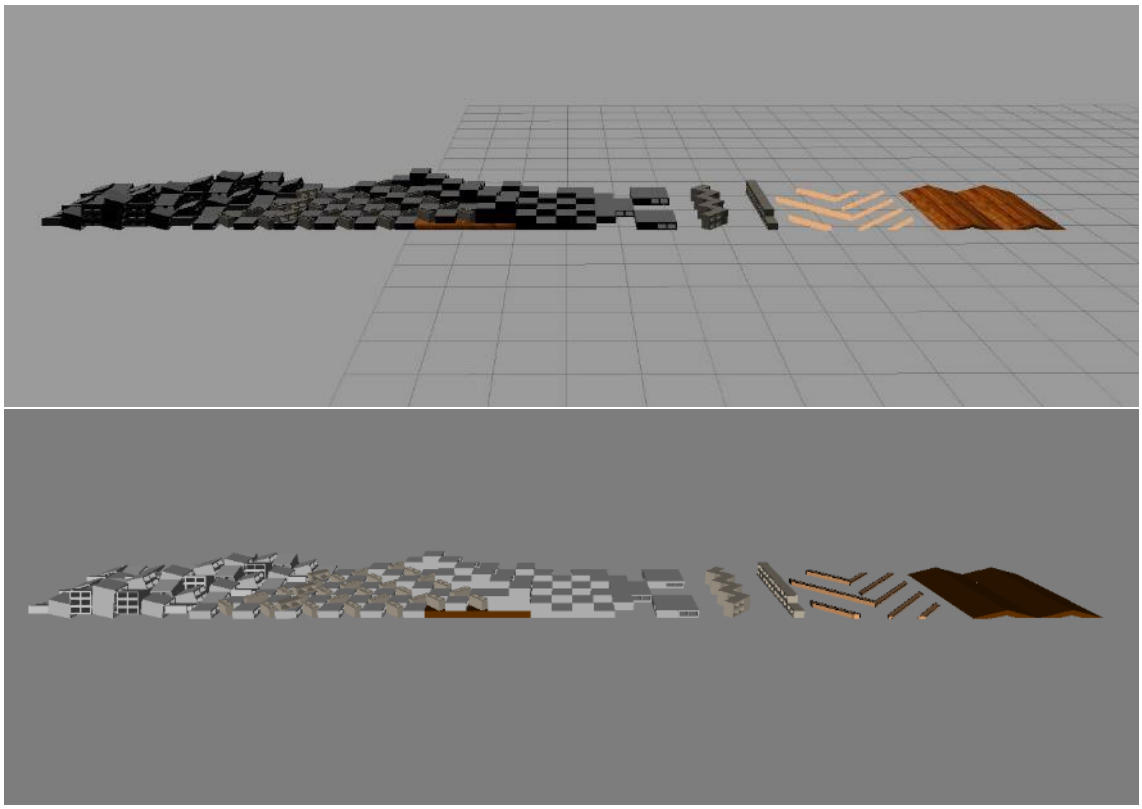

Figura 5.4. Entorno de la tarea 2 en Gazebo (parte superior) y OpenRAVE (parte inferior) sin el robot Atlas

El fichero *drc_practice_task_2.launch* que carga el entorno de la tarea 2 en Gazebo no funciona correctamente, por lo que no se puede ver el aspecto de este entorno con el robot en Gazebo. Vemos únicamente el entorno correspondiente a OpenRAVE para este caso.

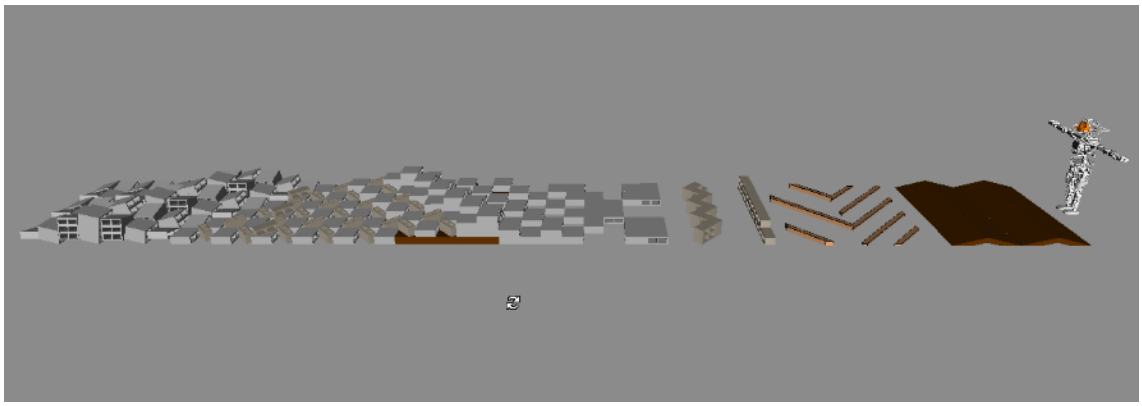

Figura 5.5. Entorno de la tarea 2 en OpenRAVE con el robot Atlas

5.2.3 Tarea 5

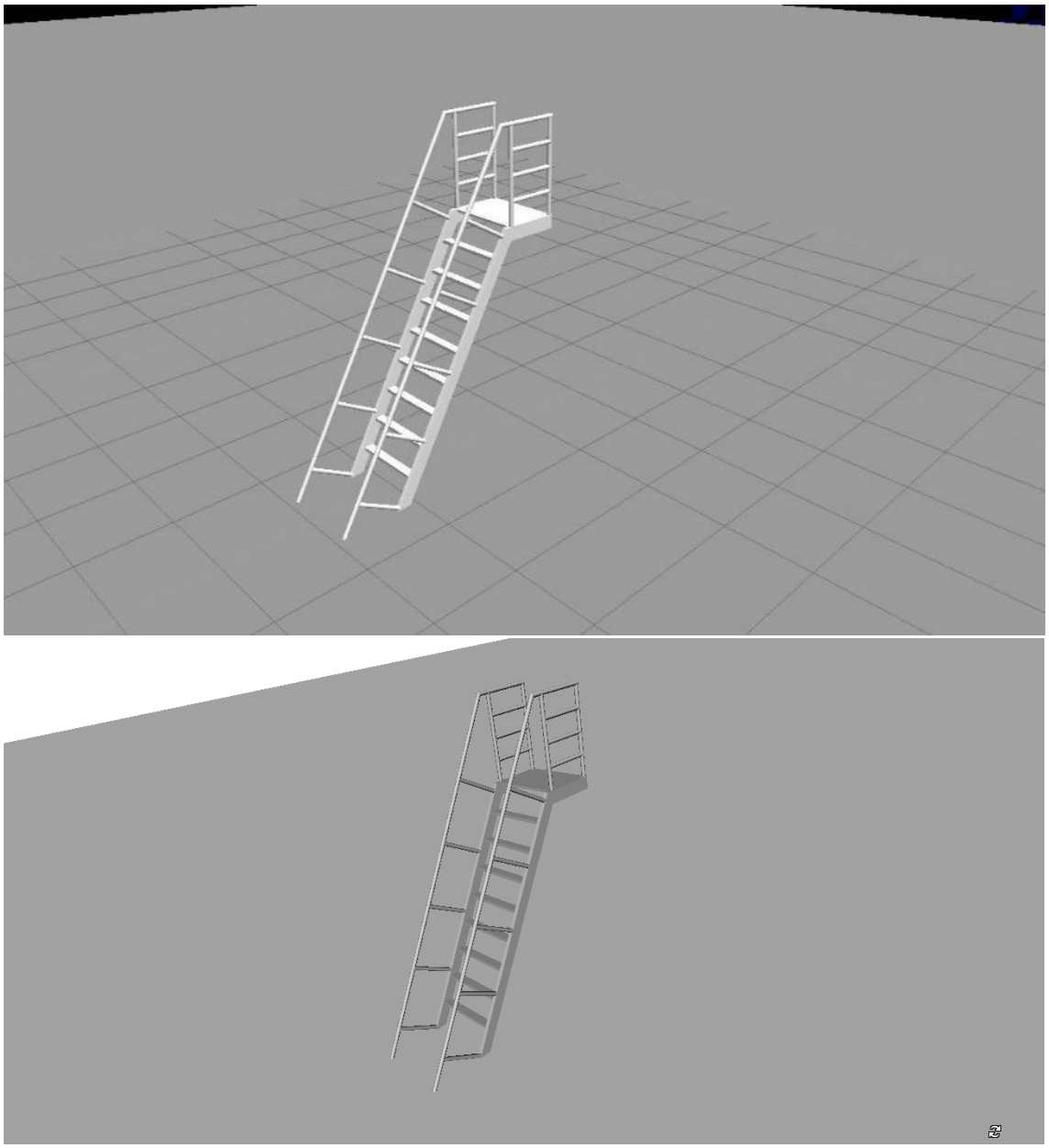

Figura 5.6. Entorno de la tarea 5 en Gazebo (parte superior) y OpenRAVE (parte inferior) sin el robot Atlas

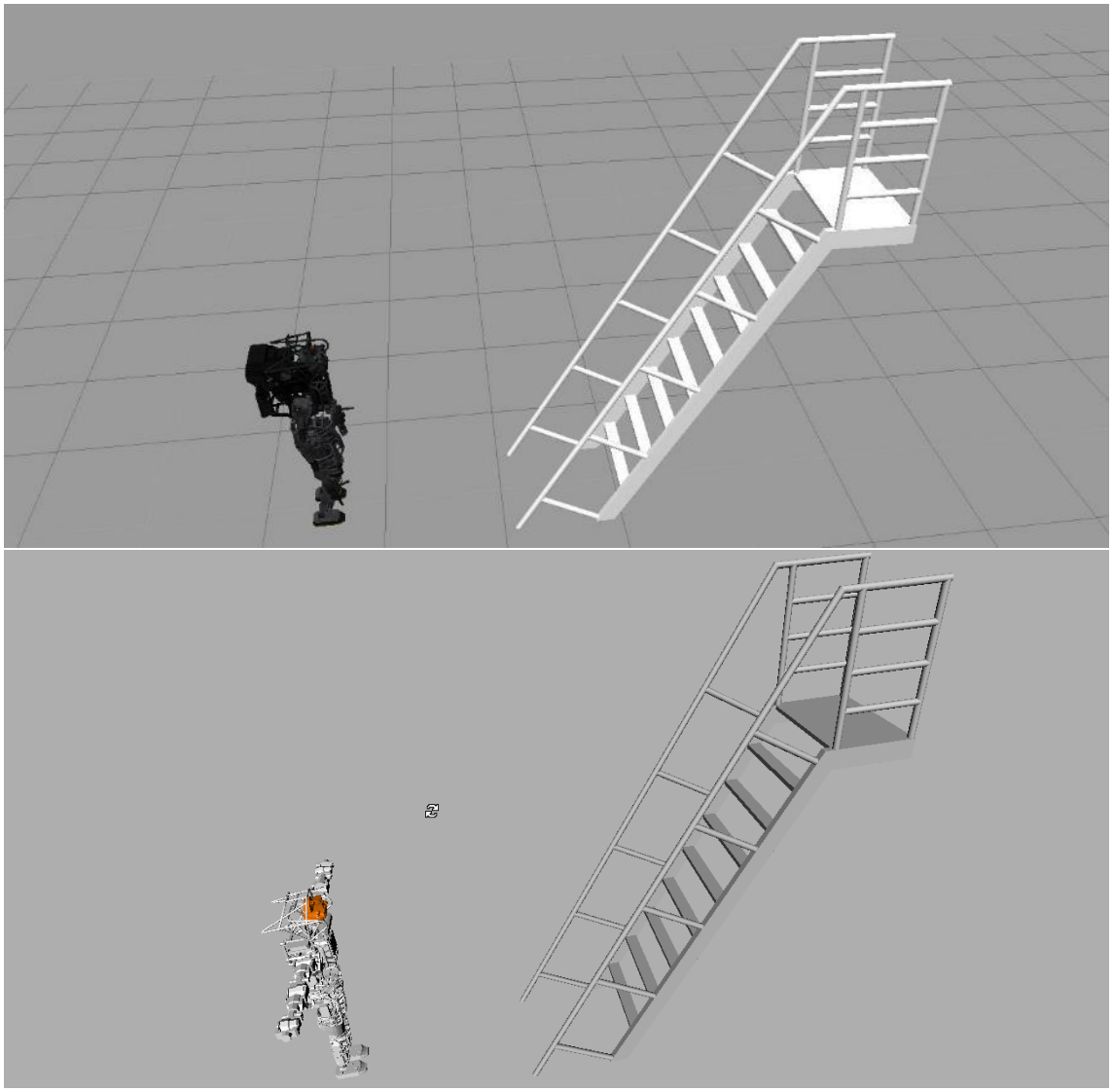

Figura 5.7. Entorno de la tarea 5 en Gazebo (parte superior) y en OpenRAVE (parte inferior) con el robot Atlas

5.3 VEHÍCULO

La Figura 5.8 muestra el aspecto del vehículo en el entorno de la tarea 1 en Gazebo y del mismo adaptado para OpenRAVE.

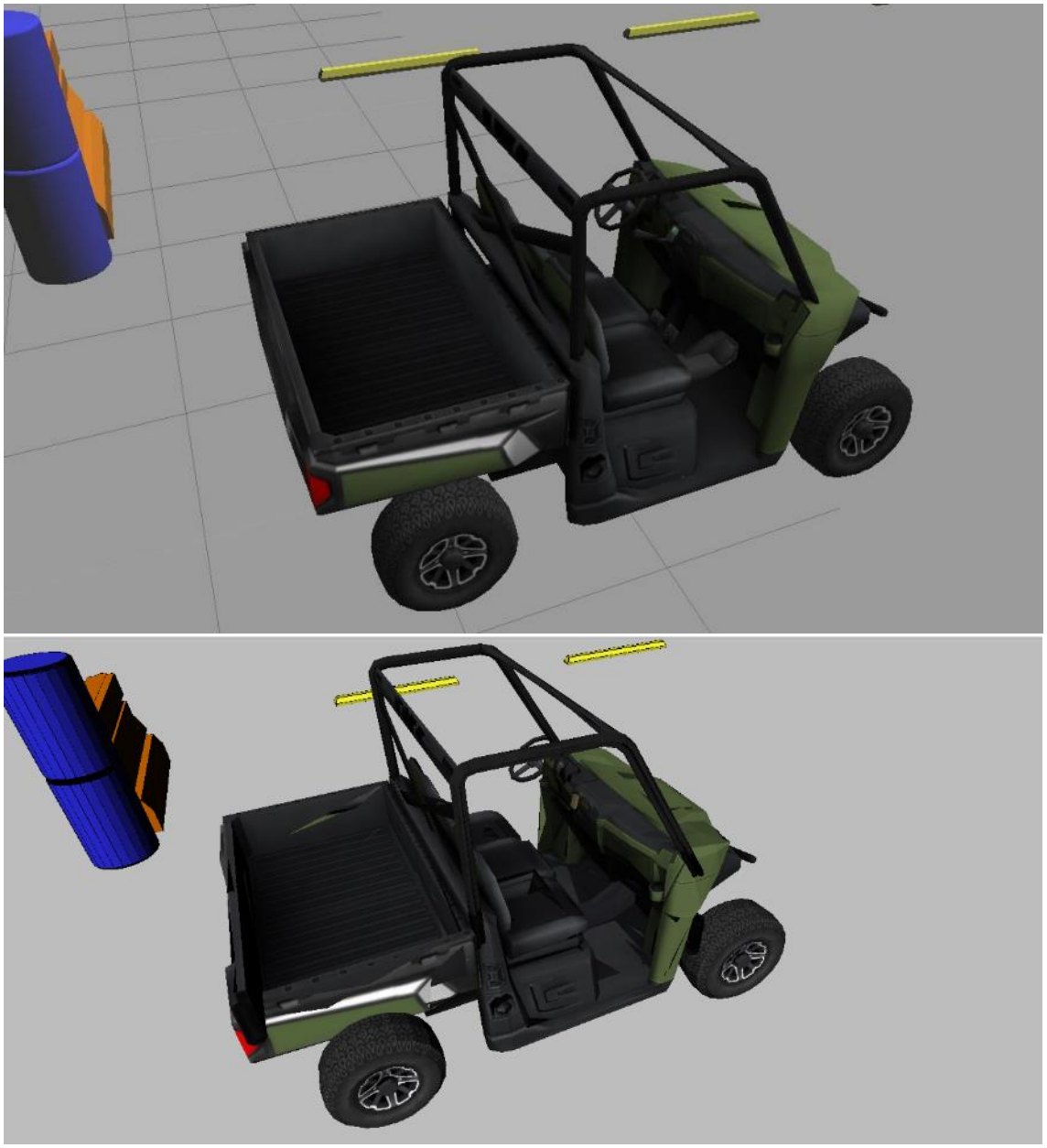

Figura 5.8. Representación del vehículo en el entorno de la tarea 1 en Gazebo (superior) y OpenRAVE (inferior)

Tanto el robot como el vehículo es posible cargarlos directamente en el código del entorno o mediante Python al lanzar el simulador.

5.4 ROBOT ATLAS

Como resultado de la conversión utilizando la herramienta `collada_urdf` se obtiene un robot completamente funcional y que cumple los objetivos, aunque se pierden algunas geometrías y las texturas asociadas al robot en Gazebo.

Al igual que pasa con el vehículo, es posible cargar el robot directamente en el mismo archivo del entorno, incluyendo una referencia al robot en dicho fichero, como se muestra en el Fragmento de código 5.9. Otra opción es añadir el robot al entorno haciendo uso del lenguaje Python, mediante la instrucción correspondiente en el fichero **.py*, como se ha visto al principio de este capítulo.

Si se opta por la segunda opción, es necesario crear un fichero **.robot.xml* distinto por cada entorno, que sitúe al robot en la posición correspondiente a cada escenario. La posición del robot en el entorno viene dada en el archivo **.launch* asociado al entorno en Gazebo.

Por ejemplo, para situar el robot en el entorno de la tarea 1, se parte del archivo *drc_practice_task_1.launch*, que indica la traslación y rotación del robot en el entorno en el Fragmento de código 5.8, y se crea el fichero *atlas_task_1.robot.xml*, cuyo código aparece en el Fragmento de código 5.9, que carga y sitúa el robot en el entorno según la transformación dada.

```
<!-- initial robot position -->
<param name="robot_initial_pose/x"     value="3.0" type="double"/>
<param name="robot_initial_pose/y"     value="0" type="double"/>
<param name="robot_initial_pose/z"     value="1.0" type="double"/>
<param name="robot_initial_pose/roll"  value="0" type="double"/>
<param name="robot_initial_pose/pitch" value="0" type="double"/>
<param name="robot_initial_pose/yaw"   value="0" type="double"/>
```

Fragmento de código 5.8. Posición inicial del robot en el entorno de la tarea 1 en Gazebo

```
<Robot name="atlas" file="atlas.dae">
    <Translation>3 0 1</Translation>
</Robot>
```

Fragmento de código 5.9. Carga del robot en la posición inicial en el entorno de la tarea 1 en OpenRAVE

El resultado es que el robot Atlas aparece colocado en la misma posición inicial en los ficheros de Gazebo y OpenRAVE, como se aprecia en las imágenes anteriores de los entornos.

A continuación, en la Figura 5.9, se muestra el resultado de cargar Atlas en un entorno vacío en Gazebo y en OpenRAVE.

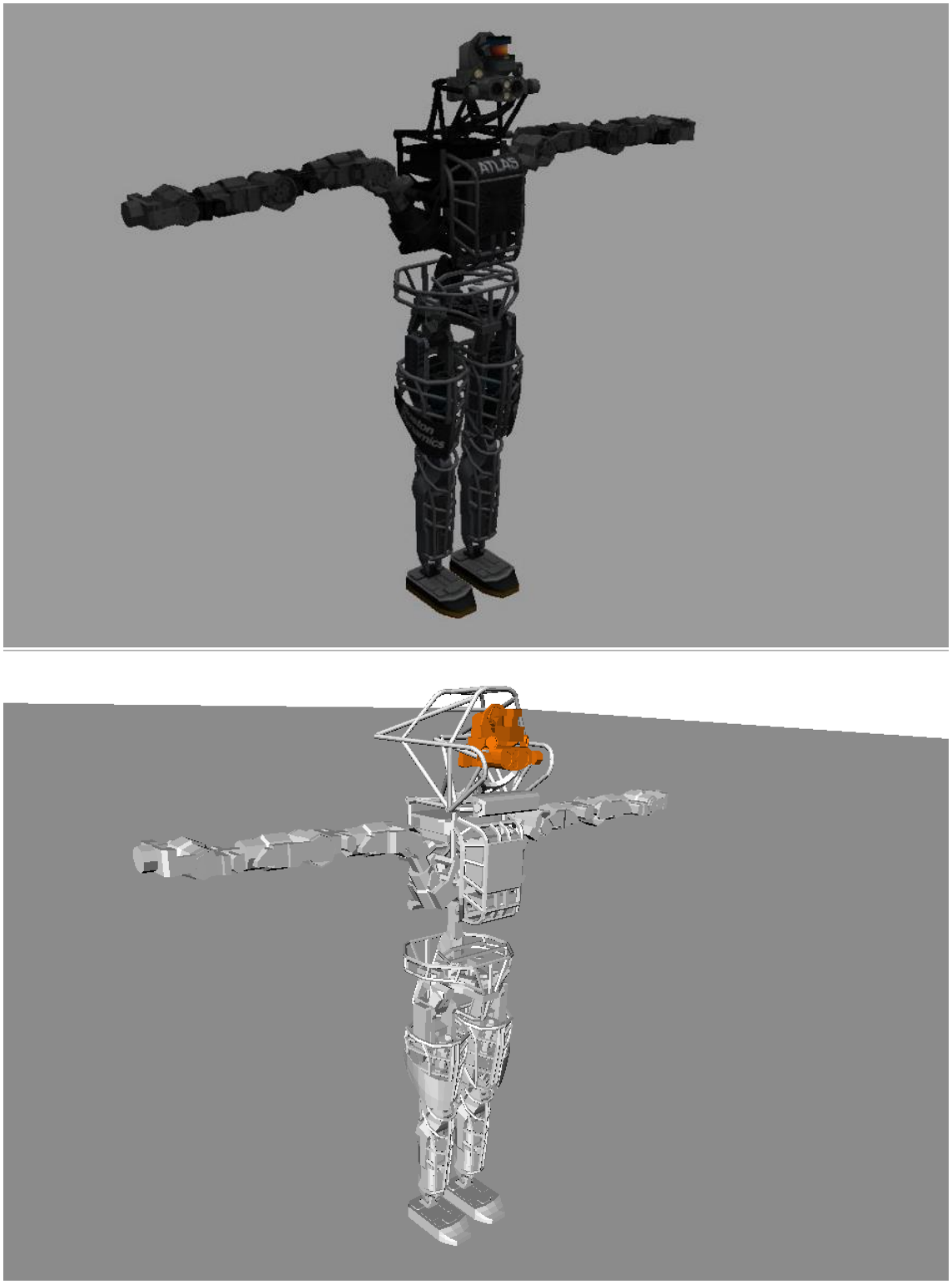

Figura 5.9. Atlas en un entorno vacío en Gazebo (superior) y OpenRAVE (inferior)

Como ya se ha comentado, se aprecian diferencias en las texturas y algunas geometrías, que no afectan a la funcionalidad.

6 CONCLUSIONES Y LÍNEAS FUTURAS DE TRABAJO

6.1 CONCLUSIONES

Como ya se ha visto en el Capítulo 5, en que se recogen los resultados del proyecto, se ha cumplido con el objetivo inicial del mismo: adaptar para su uso en OpenRAVE determinados entornos de DRC creados para Gazebo.

Como resultado del análisis llevado a cabo para la adaptación de los entornos, se ha podido recopilar información sobre la creación de entornos en Gazebo y OpenRAVE. Esta información, aunque se ha orientado a los entornos de DRC estudiados en el proyecto, se puede generalizar para la conversión de otros entornos. Por tanto, el proyecto proporciona una guía para la adaptación de entornos entre Gazebo y OpenRAVE.

Al realizar el análisis, se ha comprobado que no siempre es posible encontrar una correspondencia exacta para la definición de todos los elementos y características de un entorno en distintas plataformas, debido a las diferencias que existen en el funcionamiento de distintos programas. Un programa de simulación es una herramienta que permite modelar la realidad. Como tal, a la hora de definir cómo se va a realizar esta adaptación del mundo real a un ámbito virtual, es necesario buscar un compromiso entre eficiencia y precisión. Cada plataforma de simulación entiende y ejecuta estas aproximaciones de manera diferente. Es por esto que se dan diferencias en los procesos de creación de entornos y en ocasiones no hay correspondencia directa de un formato en otro, son limitaciones que vienen dadas por las características de cada simulador.

6.2 APORTACIONES

Las aportaciones de este proyecto son varias.

La primera, específica de este proyecto y dirigida a un objetivo concreto, es la adaptación de los entornos y del robot Atlas de la competición DRC de Gazebo a OpenRAVE. Se ha obtenido un robot funcional que puede utilizarse en simulaciones en los entornos.

Como aportación más general, se ha realizado un análisis detallado del proceso de creación de entornos tanto en Gazebo como en OpenRAVE. Se ha recopilado y ampliado la documentación existente. Para ello se ha recurrido, además de a la documentación oficial publicada en la web de cada simulador, a los archivos del código fuente de los programas, foros de resolución de problemas y realización de pruebas con los simuladores. Los resultados de este estudio se resumen en la Tabla 3.22 al final del Capítulo 3.

Finalmente, a partir de este análisis, se han desarrollado las pautas a seguir para la adaptación de entornos y elementos robóticos de Gazebo a OpenRAVE. Este proyecto sirve como guía para posibles futuras conversiones.

6.3 LÍNEAS FUTURAS DE TRABAJO

En este apartado se va a reflexionar sobre potenciales mejoras al proyecto y posibilidades de desarrollo que presenta el trabajo realizado.

Algunas de las mejoras posibles son:

- Realizar la adaptación a OpenRAVE del resto de entornos de DRC para trabajar en el desarrollo de otras habilidades del robot, como manipulación, destreza, toma de decisiones o fuerza.
- Conversión manual del robot Atlas para incluir las texturas y geometrías que se pierden al realizar la conversión automática y que quede estéticamente más realista. Siguiendo las pautas dadas en este documento es posible adaptar de manera manual el robot Atlas al formato XML propio de OpenRAVE, partiendo del robot descrito en los archivos de DRCSim. Otra alternativa es modificar y completar el archivo del robot Atlas generado automáticamente en formato COLLADA con la información relativa a las texturas y geometrías que faltan. Sin embargo, para este proceso sería necesario un nuevo análisis, ya que el formato COLLADA para definición de robots en OpenRAVE no se ha estudiado en este proyecto.
- Describir el proceso de adaptación inverso: conversión de entornos creados para OpenRAVE al formato propio de Gazebo. Partiendo de la información recogida en este documento, este análisis debería de resultar relativamente sencillo.
- En último término, se podría estudiar la posibilidad de adaptación de entornos y robots para otras plataformas de simulación robótica.

Como continuación y tomando como base este proyecto, se podría desarrollar un software para realizar la conversión de entornos y elementos robóticos de manera automática de Gazebo a OpenRAVE. Hay que tener en cuenta, sin embargo, que la elaboración de una aplicación de este tipo resulta un proyecto de gran envergadura ya que, más allá de las características básicas de un entorno, que pueden ser relativamente fáciles de adaptar ya que existe una correspondencia clara entre los programas, no siempre resulta fácil encontrar esta relación, hay excepciones a las reglas generales, y ambos programas son de una gran complejidad, por lo que resulta muy complicado tener en cuenta todas las posibilidades.

REFERENCIAS

[1] S. Ivaldi, V. Padois y F. Nori, «Tools for dynamics simulation of robots: a survey,» 27 febrero 2014. [En línea]. Disponible: http://arxiv.org/pdf/1402.7050.pdf. [Último acceso: 2 diciembre 2014].

[2] «The ARGoS Website,» 2013. [En línea]. Disponible: http://iridia.ulb.ac.be/argos/home.php. [Último acceso: 2 diciembre 2014].

[3] Coppelia Robotics, «Coppelia Robotics v-rep: Create. Compose. Simulate. Any Robot,» 1 octubre 2014. [En línea]. Disponible: http://www.coppeliarobotics.com/. [Último acceso: 2 diciembre 2014].

[4] Cyberbotics Ltd., «Webots: robot simulator,» 2 diciembre 2014. [En línea]. Disponible: http://www.cyberbotics.com/. [Último acceso: 2 diciembre 2014].

[5] «Robotran - Home,» 2014. [En línea]. Disponible: http://www.robotran.be/. [Último acceso: 2 diciembre 2014].

[6] DARPA, «DRC,» 2014. [En línea]. Disponible: http://www.theroboticschallenge.org/. [Último acceso: 10 noviembre 2014].

[7] DARPA, «DARPA Robotics Challenge,» 2014. [En línea]. Disponible: http://www.darpa.mil/Our_Work/TTO/Programs/DARPA_Robotics_Challenge.aspx. [Último acceso: 10 noviembre 2014].

[8] DARPA, «DRC Final Announcements | DRC,» 2014. [En línea]. Disponible: http://www.theroboticschallenge.org/content/drc-finals-announcement. [Último acceso: 11 noviembre 2014].

[9] DARPA, «DRC Trials 2013 Countdown: A Look at the Competition Course,» 16 diciembre 2013. [En línea]. Disponible: http://www.darpa.mil/NewsEvents/Releases/2013/12/16.aspx. [Último acceso: 8 octubre 2014].

[10] A. Jacoff, R. Sheh, A. Virts, D. Schmitt, A. Moore, J. Quinby y J. Pippine, «DARPA Robotics Challenge DRC Trials 2013 Initial Task Descriptions,» 2013. [En línea]. Disponible: http://theroboticschallenge.org/local/documents/DRC%20Trials%20Initial%20Task%20Descriptions%20DISTAR%20Case%2021473.pdf. [Último acceso: 6 agosto 2013].

[11] DARPA, «Members of top nine software teams move forward from DARPA's Virtual Robotics Challenge,» 27 junio 2013. [En línea]. Disponible: http://www.darpa.mil/NewsEvents/Releases/2013/06/27.aspx. [Último acceso: 11 noviembre 2014].

[12] «osrf / drcsim — Bitbucket,» 2014. [En línea]. Disponible: https://bitbucket.org/osrf/drcsim. [Último acceso: 11 noviembre 2014].

[13] Boston Dynamics, «Boston Dynamics: Dedicated to the Science and Art of How Things Move,» 2013. [En línea]. Disponible:

http://www.bostondynamics.com/robot_Atlas.html. [Último acceso: 11 noviembre 2014].

[14] DARPA, «DRC Trials 2013 Countdown: Anatomy of a Disaster-Response Robot,» 19 diciembre 2013. [En línea]. Disponible: http://www.darpa.mil/NewsEvents/Releases/2013/12/19.aspx. [Último acceso: 11 noviembre 2014].

[15] OSRF, «Projects | Open Source Robotics Foundation,» 2012. [En línea]. Disponible: http://www.osrfoundation.org/projects. [Último acceso: 10 octubre 2014].

[16] OSRF, «Gazebo,» 2014. [En línea]. Disponible: http://www.gazebosim.org/. [Último acceso: 10 octubre 2014].

[17] «OpenRAVE | Overview | OpenRAVE Documentation,» 2014. [En línea]. Disponible: http://openrave.org/docs/latest_stable/overview/. [Último acceso: 10 octubre 2014].

[18] «OpenRAVE | Welcome to Open Robotics Automation Virtual Environment | OpenRAVE Documentation,» 2014. [En línea]. Disponible: http://openrave.org/docs/latest_stable/. [Último acceso: 10 octubre 2014].

[19] R. Diankov, «Automated Construction of Robotic Manipulation Programs,» 2010.

[20] «OpenRAVE | SourceForge.net,» 2013. [En línea]. Disponible: http://sourceforge.net/projects/openrave/. [Último acceso: 10 octubre 2014].

[21] R. Diankov, «Rosen Diankov | Research and Projects,» 29 octubre 2013. [En línea]. Disponible: http://www.programmingvision.com/research.html. [Último acceso: 10 octubre 2014].

[22] «viewer qtcoin,» [En línea]. Disponible: http://openrave.org/docs/latest_stable/interface_types/viewer/qtcoin/. [Último acceso: 12 noviembre 2014].

[23] OSRF, «Gazebo Components,» 2014. [En línea]. Disponible: http://www.gazebosim.org/tutorials?tut=components&cat=get_started. [Último acceso: 15 octubre 2014].

[24] «World File Syntax - previous version,» 4 agosto 2007. [En línea]. Disponible: http://playerstage.sourceforge.net/doc/Gazebo-manual-0.8.0-pre1-html/config_syntax.html. [Último acceso: 20 octubre 2014].

[25] OpenRAVE, «Robot Concepts 0.9.0,» 18 marzo 2013. [En línea]. Disponible: http://openrave.org/docs/latest_stable/coreapihtml/arch_robot.html. [Último acceso: 20 octubre 2014].

[26] OSRF, «Create Model,» 2014. [En línea]. Disponible: http://www.gazebosim.org/tutorials?tut=build_model&cat=build_robot. [Último acceso: 20 octubre 2014].

[27] W3C, «Extensible Markup Language (XML),» 2002. [En línea]. Disponible: http://www.w3.org/XML/. [Último acceso: 10 octubre 2014].

[28] L. Quin, «W3C Working Draft 14-Nov-96,» W3C, 14 noviembre 1996. [En línea]. Disponible: http://www.w3.org/TR/WD-xml-961114.html. [Último acceso: 18 noviembre 2014].

[29] B. Bos, «XML in 10 points,» W3C, 13 noviembre 2001. [En línea]. Disponible: http://www.w3.org/XML/1999/XML-in-10-points. [Último acceso: 18 noviembre 2014].

[30] OSRF, «URDF in Gazebo,» 2014. [En línea]. Disponible: http://gazebosim.org/tutorials/?tut=ros_urdf. [Último acceso: 10 octubre 2014].

[31] OSRF, «ROS/Patterns/RobotModelling - ROS Wiki,» 7 julio 2014. [En línea]. Disponible: http://wiki.ros.org/ROS/Patterns/RobotModelling. [Último acceso: 10 octubre 2014].

[32] «osrf / sdformat — Bitbucket,» 2014. [En línea]. Disponible: https://bitbucket.org/osrf/sdformat. [Último acceso: 10 octubre 2014].

[33] OSRF, «SDF,» 2014. [En línea]. Disponible: http://sdformat.org/. [Último acceso: 18 noviembre 2014].

[34] OSRF, «xacro - ROS Wiki,» 10 septiembre 2014. [En línea]. Disponible: http://wiki.ros.org/xacro. [Último acceso: 13 octubre 2014].

[35] OSRF, «urdf/Tutorials/Using Xacro to Clean Up a URDF File - ROS Wiki,» 19 marzo 2014. [En línea]. Disponible: http://wiki.ros.org/urdf/Tutorials/Using%20Xacro%20to%20Clean%20Up%20a%20URDF%20File. [Último acceso: 13 octubre 2014].

[36] «Format:XML - OpenRAVE,» 30 septiembre 2013. [En línea]. Disponible: http://openrave.programmingvision.com/wiki/index.php/Format:XML. [Último acceso: 18 noviembre 2014].

[37] «OpenRAVE | Robots Overview | OpenRAVE Documentation,» 2014. [En línea]. Disponible: http://openrave.org/docs/latest_stable/robots_overview/. [Último acceso: 10 octubre 2014].

[38] «OpenRAVE | COLLADA Robot Extensions (Version 0.3.3) | OpenRAVE Documentation,» 2014. [En línea]. Disponible: http://openrave.org/docs/latest_stable/collada_robot_extensions/. [Último acceso: 10 octubre 2014].

[39] OSRF, «collada_urdf - ROS Wiki,» 9 mayo 2014. [En línea]. Disponible: http://wiki.ros.org/collada_urdf. [Último acceso: 20 octubre 2014].

[40] OSRF, «Model structure and requirements,» 2014. [En línea]. Disponible: http://www.gazebosim.org/tutorials?tut=model_structure&cat=build_robot. [Último acceso: 10 noviembre 2014].

[41] «COLLADA FAQ - COLLADA,» 3 diciembre 2008. [En línea]. Disponible: https://collada.org/mediawiki/index.php/COLLADA_FAQ. [Último acceso: 10 octubre 2014].

[42] Khronos Group, «COLLADA - 3D Asset Exchange Schema,» 2014. [En línea]. Disponible: https://www.khronos.org/collada/. [Último acceso: 10 octubre 2014].

[43] Wikipedia, «STL (file format) - Wikipedia, the free encyclopedia,» 18 noviembre 2014. [En línea]. Disponible: http://en.wikipedia.org/wiki/STL_(file_format). [Último acceso: 20 noviembre 2014].

[44] SGI, «SGI - Developer Central Open Source | Open Inventor,» 2012. [En línea]. Disponible: http://oss.sgi.com/projects/inventor/. [Último acceso: 20 octubre 2014].

[45] Web3D Consortium, «X3D & VRML, The Most Widely Used 3D Formats | Web3D Consortium,» 2014. [En línea]. Disponible: http://www.web3d.org/x3d-vrml-most-widely-used-3d-formats. [Último acceso: 23 noviembre 2014].

[46] R. Smith, «Open Dynamics Engine,» 23 febrero 2006. [En línea]. Disponible: http://www.ode.org/ode-latest-userguide.html. [Último acceso: 24 noviembre 2014].

[47] T. Moulard, «gazebo-deb/joint.sdf at master · thomas-moulard/gazebo-deb · GitHub,» 15 abril 2013. [En línea]. Disponible: https://github.com/thomas-moulard/gazebo-deb/blob/master/gazebo/sdf/1.3/joint.sdf. [Último acceso: 7 octubre 2014].

[48] S. Peters, «osrf / sdformat / Pull request #83: SDF 1.5: add flag to fix joint axis frame #43 (gazebo issue 494) — Bitbucket,» 19 febrero 2014. [En línea]. Disponible: https://bitbucket.org/osrf/sdformat/pull-request/83/sdf-15-add-flag-to-fix-joint-axis-frame-43/diff. [Último acceso: 25 noviembre 2014].

[49] OSRF, «Tutorial: Using a URDF in Gazebo,» 2014. [En línea]. Disponible: Tutorial: Using a URDF in Gazebo. [Último acceso: 10 octubre 2014].

[50] «OpenRAVE | Conventions and Guidelines | OpenRAVE Documentation,» 2014. [En línea]. Disponible: http://openrave.org/docs/latest_stable/geometric_conventions/. [Último acceso: 23 noviembre 2014].

[51] «OpenRAVE::SensorBase Class Reference 0.9.0,» 18 marzo 2013. [En línea]. Disponible: http://openrave.org/docs/latest_stable/coreapihtml/classOpenRAVE_1_1SensorBase.html. [Último acceso: 2 diciembre 2014].

[52] R. Diankov, «openrave/plugins/basesensors at 7d0dc9c945bab4fc6e3a703f707bb64656b7b897 · rdiankov/openrave · GitHub,» 7 mayo 2014. [En línea]. Disponible: https://github.com/rdiankov/openrave/tree/7d0dc9c945bab4fc6e3a703f707bb64656b7b897/plugins/basesensors. [Último acceso: 2 diciembre 2014].

[53] R. Diankov y C. Ferraro, «OpenRAVE Users List - force sensor,» 3 mayo 2011. [En línea]. Disponible: http://openrave-users-list.185357.n3.nabble.com/force-sensor-td2894578.html. [Último acceso: 2 diciembre 2014].

[54] «Physics engine - Wikipedia, the free encyclopedia,» 2 octubre 2014. [En línea]. Disponible: http://en.wikipedia.org/wiki/Physics_engine. [Último acceso: 20 octubre 2014].

[55] S. Peters y J. Hsu, «Comparison of Rigid Body Dynamic Simulators for Robotic Simulation in Gazebo,» de *ROS Developer Conference*, Chicago, Illinois, USA, 2014.

[56] «OpenRAVE | Interface Types | OpenRAVE Documentation,» 2014. [En línea]. Disponible: http://openrave.org/docs/latest_stable/interface_types/#physicsengine. [Último acceso: 10 octubre 2014].

[57] R. Smith, «Open Dynamics Engine - home,» 28 mayo 2007. [En línea]. Disponible: http://www.ode.org/. [Último acceso: 10 octubre 2014].

[58] «Products that use ODE - ODE Wiki,» 21 enero 2013. [En línea]. Disponible: http://ode-wiki.org/wiki/index.php?title=Products_that_use_ODE#Robotics_simulators. [Último acceso: 10 octubre 2014].

[59] «Open Dynamics Engine: Bodies,» 2009. [En línea]. Disponible: http://opende.sourceforge.net/docs/group_bodies.html. [Último acceso: 10 octubre 2014].

[60] OSRF, «SDF,» 2014. [En línea]. Disponible: http://sdformat.org/spec. [Último acceso: 20 noviembre 2014].

[61] «OpenRAVE | ode - oderave | OpenRAVE Documentation,» 2014. [En línea]. Disponible: http://openrave.org/docs/latest_stable/interface_types/physicsengine/ode/. [Último acceso: 10 octubre 2014].

[62] Trimble Navigation Limited, «The SketchUp Story | SketchUp,» 2013. [En línea]. Disponible: http://www.sketchup.com/about/sketchup-story. [Último acceso: 28 noviembre 2014].

[63] Blender Foundation, «blender.org - Home of the Blender project - Free and Open 3D Creation Software,» 2014. [En línea]. Disponible: http://www.blender.org/. [Último acceso: 29 noviembre 2014].

[64] «Extensions:2.4/Py/Scripts/Export/IV-OpenInventor - BlenderWiki,» 2013. [En línea]. Disponible: http://wiki.blender.org/index.php/Extensions:2.4/Py/Scripts/Export/IV-OpenInventor. [Último acceso: 9 septiembre 2014].

[65] N. Letwory, «Importers and Exporters | Blender Code,» 19 enero 2011. [En línea]. Disponible: http://code.blender.org/index.php/2011/01/importers-and-exporters/. [Último acceso: 9 septiembre 2014].

[66] N. Koenig, «osrf / gazebo / Pull request #392: Allow submeshes to be used as visuals and collisions — Bitbucket,» 20 marzo 2013. [En línea]. Disponible: https://bitbucket.org/osrf/gazebo/pull-request/392/allow-submeshes-to-be-used-as-visuals-and/diff#chg-gazebo/sdf/1.4/geometry.sdf. [Último acceso: 20 octubre 2014].

[67] OSRF, «DRC/UserGuide - Gazebo Wiki,» 3 junio 2013. [En línea]. Disponible: http://gazebosim.org/wiki/DRC/UserGuide. [Último acceso: 29 mayo 2014].

[68] Python Software Foundation, «About Python™ | Python.org,» 2014. [En línea]. Disponible: https://www.python.org/about/. [Último acceso: 1 diciembre 2014].

[69] «OpenRAVE | Quick examples in openravepy | OpenRAVE Documentation,» 2014. [En línea]. Disponible:

http://openrave.org/docs/latest_stable/tutorials/openravepy_examples/. [Último acceso: 15 abril 2014].

[70] «OpenRAVE | tutorial_grasptransform Module | OpenRAVE Documentation,» 2014. [En línea]. Disponible: http://openrave.org/docs/0.8.0/openravepy/examples.tutorial_grasptransform/. [Último acceso: 10 noviembre 2014].

[71] K. Hauser, «KrisLibrary/geometry/PQP at master · krishauser/KrisLibrary · GitHub,» 4 julio 2014. [En línea]. Disponible: https://github.com/krishauser/KrisLibrary/tree/master/geometry/PQP. [Último acceso: 10 octubre 2014].

[72] Silicon Graphics International Corp., «SGI - The Trusted Leader in High Performance Computing: Compute, Data Management, Data Analytics,» 2014. [En línea]. Disponible: http://www.sgi.com/. [Último acceso: 23 noviembre 2014].

[73] «ROS.org | Powering the world's robots,» 2014. [En línea]. Disponible: http://www.ros.org/. [Último acceso: 10 octubre 2014].

[74] DARPA, «Our Work,» 2014. [En línea]. Disponible: http://www.darpa.mil/Our_Work/. [Último acceso: 10 Noviembre 2014].

[75] GRASP, «OpenRAVE plugins — OpenGRASP,» 21 noviembre 2011. [En línea]. Disponible: http://opengrasp.sourceforge.net/openravePlugins.html. [Último acceso: 30 noviembre 2014].

[76] OSRF, «Open Source Robotics Foundation,» 2014. [En línea]. Disponible: http://www.osrfoundation.org/. [Último acceso: 11 noviembre 2014].